PB84-143932

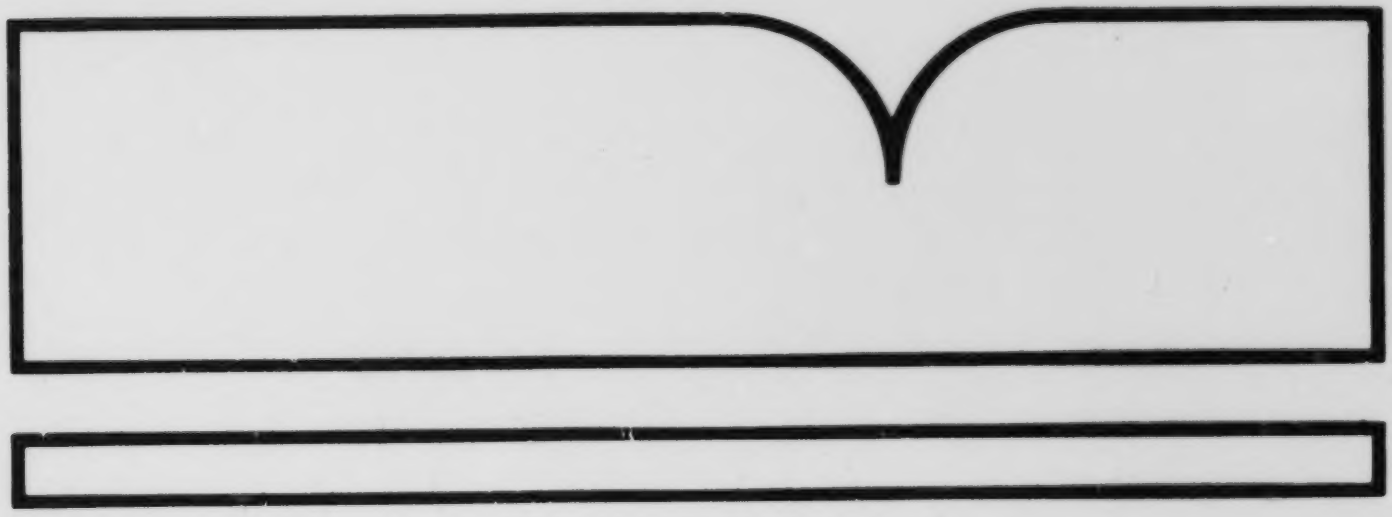
# Water Supply Simulation Model
# Volume 3. Documentation

Gates (W. E.) and Associates, Inc., Batavia, OH

Prepared for

Municipal Environmental Research Lab.
Cincinnati, OH

Nov 83



U.S. Department of Commerce
National Technical Information Service
NTIS

PB84-143932

EPA-600/2-83-121c
November 1983

# WATER SUPPLY SIMULATION MODEL

## VOLUME III - DOCUMENTATION

by

Paul Wehrman and Richard Males
W. E. Gates and Associates, Inc.
Batavia, Ohio 45103

Robert M. Clark
Drinking Water Research Division Municipal
Environmental Research Laboratory
U. S. Environmental Protection Agency
Cincinnati, OH 45268

Contract No. 68-03-2941

Project Officer

Robert M. Clark
Drinking Water Research Division
Municipal Environmental Research Laboratory
Cincinnati, Ohio 45268

REPRODUCED BY
NATIONAL TECHNICAL
INFORMATION SERVICE
U.S. DEPARTMENT OF COMMERCE
SPRINGFIELD, VA. 22161

MUNICIPAL ENVIRONMENTAL PROTECTION AGENCY
OFFICE OF RESEARCH AND DEVELOPMENT
U.S. ENVIRONMENTAL PROTECTION AGENCY
CINCINNATI, OH 45268

TECHNICAL REPORT DATA
*(Please read Instructions on the reverse before completing)*

| 1. REPORT NO. | 2. | 3. RECIPIENT'S ACCESSION NO. |
|---|---|---|
| EPA-600/2-83-121c | | PB84 143932 |

| 4. TITLE AND SUBTITLE | 5. REPORT DATE |
|---|---|
| WATER SUPPLY SIMULATION MODEL: Volume III. Documentation | November 1983 |
| | 6. PERFORMING ORGANIZATION CODE |
| 7. AUTHOR(S) | 8. PERFORMING ORGANIZATION REPORT NO. |
| Paul Wehrman, Richard Males, and Robert M. Clark* | |
| 9. PERFORMING ORGANIZATION NAME AND ADDRESS | 10. PROGRAM ELEMENT NO. |
| W. E. Gates & Associates<br>Batavia, Ohio 45103<br>*USEPA, Cincinnati, OH 45268 | 11. CONTRACT/GRANT NO.<br>Contract 68-03-2941 |
| 12. SPONSORING AGENCY NAME AND ADDRESS | 13. TYPE OF REPORT AND PERIOD COVERED |
| Municipal Environmental Research Laboratory - Cin., OH<br>Office of Research and Development<br>U.S. Environmental Protection Agency<br>Cincinnati, OH 45268 | 14. SPONSORING AGENCY CODE<br>EPA/600/14 |

15. SUPPLEMENTARY NOTES

Project officer: Robert M. Clark (513)684-7488

16. ABSTRACT

This three-volume report describes the development of a water supply simulation model (WSSM), a system of computer programs that allows for a systematic evaluation of the physical and economic characteristics of a water distribution system in a spatial framework.

The WSSM concept views a water utility as a network overlayed upon a spatial distribution of supply and demand. The model explicitly deals with the relationship of delivered water costs to the service requirements of spatially distributed demand. Once an adequate hydraulic simulation has been made, the model can be used to determine costs, travel time, and contaminant concentrations at various points in the network.

The model has been calibrated and tested on several water supply systems including a small utility in New Vienna, Ohio, and the Kenton County Water District No. 1 in Kenton County, Kentucky. Volume 1 of this study describes the development of the model and its underlying principals. Volume 2 discusses some of the engineering and economic concepts used in developing the model. Volume 3 is a users manual for the model.

17. KEY WORDS AND DOCUMENT ANALYSIS

| a. DESCRIPTORS | b. IDENTIFIERS/OPEN ENDED TERMS | c. COSATI Field/Group |
|---|---|---|
| | | |

| 18. DISTRIBUTION STATEMENT | 19. SECURITY CLASS *(This Report)* | 21. NO. OF PAGES |
|---|---|---|
| | | 116 |
| | 20. SECURITY CLASS *(This page)* | 22. PRICE |

EPA Form 2220-1 (9-73)

## DISCLAIMER

The information in this document has been funded wholly or in part by the United States Environmental Protection Agency under Contract No. 68-03-2941 to W. E. Gates and Associates, Inc. It has been subject to the Agency's peer and administrative review, and it has been approved for publication as an EPA document. Mention of trade names or commerical products does not constitute endoresement or recommendation for use.

## FOREWORD

The Environmental Protection Agency was created because of increasing public and governmental concern about the dangers of pollution to the health and welfare of the American people. Noxious air, foul water, and spoiled land are tragic testimonies to the deterioration of our natural environment. The complexity of that environment and interplay among its components require a concentrated and integrated attack on the problem.

Research and development is the first step in problem solution, and it involves defining the problem, measuring its impact, and searching for solutions. The Municipal Environmental Research Laboratory develops new and improved technology and systems (1) to prevent, treat, and manage wastewater, solid and hazardous waste, and pollutant discharges from municipal and community sources, (2) to preserve and treat public drinking water supplies, and (3) to minimize the adverse economic, social, health, and aesthetic effects of pollution. This publication is a product of that research and is a most vital communications link between the research and the user community.

One of the major problems facing the U.S. Enivronmental Protection Agency is meeting the requirements of the Safe Drinking Water Act. The Act emphasizes the need to consider economics in all phases of its implementation. This report presents results from a study on the development of a water supply system simulation model. The model allows the user to evaluate the tradeoffs involved in making decisions concerning system expansion, water production and facility location. It should be useful to water supply managers and planners.

Francis T. Mayo, Director
Municipal Environmental Research
Laboratory

ABSTRACT

For the water supply industry, the Safe Drinking Water Act of 1974 (Public Law 93-523) has brought about a fundamental re-examination of the way water is handled before it is delivered to the customer. The U.S. Environmental Protection Agency (EPA), designated as responsible for establishing a program that will carry out the Act's provisions, is required to issue regulations, while state agencies are to develop water supply programs, and local governments to pursue these programs. Also, water utilities are required to monitor drinking water quality.

A primary feature of the Act is that economics must be considered before regulations are promulgated. An analysis of the regulations to be promulgated under the Act indicates that its economic effects will be minor on large water supply systems, but that there may be a potentially serious economic impact on small water supply systems due to the high unit costs generally associated with small treatment plants.

One option that might provide relief from this adverse economic impact is for a small system to join with other small systems or with a larger system to form a regional water supply. Presumably, the economies of scale associated with a regional system would benefit small system consumers. Historically, few tools and techniques have been available to assist water supply operators and planners in the analysis of these and related issues.

The effort described in this report has been specifically directed at the hydraulic and economic behavior of water distribution networks, through the development of data management, display, and simulation modeling capability for a wide variety of issues involving such networks. A flexible approach which will aid in the analysis of water supply system problems has been devised, and implemented through development of a modular set of transportable computer programs.

This report is Volume III of a three-volume series that presents detailed information concerning development of the Water Supply Simulation Model (WSSM). Volume III is the user documenatation for the model. The WSSM can be used to study some of the economic trade-offs associated with regional water supply systems, as well as many other important issues in water supply. The version of the WSSM described in this report was developed through a contract with W. E. Gates and Associates, Batavia, Ohio and Fairfax, Virginia (Contract No. 68-03-2941) and is an outgrowth of an earlier pilot implementation created by EPA's Drinking Water Research Division and the University of Cincinnati Computer Center.

TABLE OF CONTENTS


| | Page |
|---|---|
| FOREWORD | iii |
| ABSTRACT | iv |
| FIGURES | ix |
| TABLES | xi |
| ACKNOWLEDGEMENTS | xii |
| INTRODUCTION | 1 |
| User Environment | 2 |
| Handware Environment | 2 |
| Hydraulic Network Analysis Model | 3 |
| Report Format | 4 |
| PROGRAM DOCUMENTATION | |
| ERCHECK | 6 |
| ESTBLINK | 8 |
| ESTBNODE | 11 |
| FIXL | 14 |
| FIXN | 17 |
| FLEXI | 20 |
| INSELK | 23 |
| INSEND | 26 |
| LCOPYATT | 29 |
| NAMEIN | 31 |
| NCOPYATT | 32 |
| NETPLOT | 34 |
| REPORTER | 41 |

TABLE OF CONTENTS (Contd)


| | Page |
|---|---|
| SCANCOL | 44 |
| SOLVER | 47 |
| TRAVEL | 52 |
| WATINT | 54 |
| WATWSSM | 58 |
| PROGRAMMER'S GUIDE | 63 |
| File Organization | 63 |
| Data Structure | 64 |
| Logical Record Access Methodology | 64 |
| SUBPROGRAM DESCRIPTIONS | |
| ADIST | 73 |
| BCHECK | 74 |
| BRWRIT | 75 |
| CLOSE | 76 |
| COLCHK | 77 |
| COMPAR | 78 |
| COVER | 79 |
| DATIME | 80 |
| GETLK | 81 |
| GETND | 82 |
| HEAD | 83 |
| HEADL | 84 |
| INHEAD | 85 |
| INITL | 86 |
| NAME | 87 |

TABLE OF CONTENTS (Contd)


| | Page |
|---|---|
| OPEN | 88 |
| PUTLK | 89 |
| PUTLKZ | 90 |
| PUTND | 91 |
| PUTNDZ | 92 |
| UHEAD | 93 |
| ULHEAD | 94 |
| UPDATN | 95 |
| DESCRIPTION OF COMMON BLOCKS USED BY SUBPROGRAMS | 96 |
| WATSIM MODIFICATIONS | 99 |
| TRANSPORTATION ISSUES | 101 |
| REFERENCES | 103 |

FIGURES


| Number | | Page |
|---|---|---|
| 1 | ERCHECK Output | 7 |
| 2 | ESTBLINK Input Deck (Partial) | 10 |
| 3 | ESTLINK Sample Output (Partial) | 10 |
| 4 | ESTBNODE Input (Partial) | 13 |
| 5 | ESTBNODE Sample Output (Partial) | 13 |
| 6 | FIXL Sample Input | 16 |
| 7 | FIXL Sample Output | 16 |
| 8 | FIXN Sample Input | 19 |
| 9 | FIXN Sample Output | 19 |
| 10 | FLEXI Input (with Runner) | 22 |
| 11a | FLEXI OUTPUT (Logical Unit 6) | 22 |
| 11b | FLEXI Output (Logical Unit 10) | 22 |
| 12 | INSELK Sample Input (with Runner) | 25 |
| 13a | INSELK Sample Output (Logical Unit 6) | 25 |
| 13b | INSELK Sample Output (Logical Unit 10) | 25 |
| 14 | INSEND Sample Input (with Runner) | 28 |
| 15 | INSEND Sample Output (Logical Unit 6) | 28 |
| 16 | LCOPYATT Input, Background Mode | 30 |
| 17 | LCOPYATT Output | 30 |
| 18 | NCOPYATT Sample Input | 33 |
| 19 | NCOPYATT Sample Output | 33 |
| 20a | NETPLOT Sample Input | 38 |
| 20b | NETPLOT Sample Input | 38 |

FIGURES (Contd)


| Number | | Page |
|---|---|---|
| 21a | NETPLOT Printer Output | 38 |
| 21b | NETPLOT Printer Output | 38 |
| 22 | REPORTER Sample Input (with Runner) | 43 |
| 23a | REPORTER Sample Output (Logical Unit 10) | 43 |
| 23b | REPORTER Sample Output (Logical Unit 6) | 43 |
| 24 | SCANCOL Sample Input | 46 |
| 25a | SCANCOL Sample Output (Logical Unit 10) | 46 |
| 25b | SCANCOL Sample Output (Logical Unit 6) | 46 |
| 26a | SOLVER Sample Input | 51 |
| 26b | NODE Override Deck Input (Logical Unit 3) | 51 |
| 27a | SOLVER Sample Output (Logical Unit 6) | 51 |
| 27b | SOLVER Output (Logical Unit 10) | 51 |
| 28 | TRAVEL Sample Input (with Runner) | 53 |
| 29 | TRAVEL Sample Output (Partial) | 53 |
| 30 | WATINT Sample Input | 57 |
| 31 | WATINT Printer Output (Logical Unit 6) | 57 |
| 32 | WATINT Card - Image Output (Logical Unit 9) | 57 |
| 33 | WATWSSM Sample Input (with Runner) | 60 |
| 34 | WATWSSM Printer Output | 60 |

TABLES


| Number | | Page |
|---|---|---|
| 1 | Assignment of Logical Units to Program Modules. . . . | 61 |
| 2 | UPDATE Codes . . . . . | 62 |
| 3 | LINK Hearder Record . . . . . | 65 |
| 4 | Node Header Record . . . . . | 66 |
| 5 | Logical LINK Record . . . . . | 67 |
| 6 | Logical NODE Record . . . . . | 68 |
| 7 | Subprograms . . . . . | 72 |

ACKNOWLEDGEMENTS

Many people contributed to the development of the WSSM. Dr. Richard Stevie performed much of the early work on the original implementation of WSSM. Mr. A. Lee Cesario of the Denver Water Board, and Mr. Bruce Burbank of the Public Service Company of Colorado, provided advice, assistance, and access to their versions of the WATSIM hydraulic analysis model. Dr. M. Chandrashekar of the University of Waterloo, Ontario, graciously allowed the use of subroutines developed by him to be incorporated as the sparse matrix simultaneous equation solution techniques utilized in the System Solver module of the WSSM. The contributions of these individuals have been of significant benefit to the development of the WSSM.

The current version of the WSSM, as presented in this report, was developed by W. E. Gates and Associates, Batavia, Ohio, under Contract No. 68-03-2041. Dr. Richard Males served as principal investigator, Mr. Paul Wehrman performed much of the systems analysis and programming, and Dr. William Gates served as senior project advisor. Dr. Robert M. Clark served as project officer for USEPA. The authors would like to acknowledge the assistance of Ms Patricia Pierson in preparing this manuscript.

# INTRODUCTION

The Water Supply Simulation Model (WSSM) is a set of computer programs that assist in the organization, analysis, and display of data that is referenced to a water supply distribution network. The capabilities of the WSSM have been described in detail in Volume I of this report series.[1] Details of other supporting research activities are given in Volume II.[2] The purpose of this document is to present user and programmer-level documentation of the files, programs, and subroutines of WSSM. User-level documentation consists of a description of the purpose of each program, its restrictions, an overview of its use, format of input, and example output and program messages. Programmer-level documentation consists of a description of the data file formats, use and calling sequences of the general-purpose subroutines available within WSSM, and detailed description of the file access routines. This document is intended as a reference and is not tutorial in nature.

Major development work on the the WSSM took place on a UNIVAC system, with the final version developed on the University of Michigan's MTS (Michigan Terminal System), operating on an Amdahl computer. WSSM is programmed in "near-standard" FORTRAN IV. Issues of transportability are discussed later in this volume.

In view of the wide variety of operating environments and computer operating systems under which the WSSM may be implemented, no attempt is made to describe the job control language, which will of course be installation-specific. Rather, for each program, the files that are expected as input and output on each logical unit are defined. This information is

sufficient to allow a system programmer at any installation to develop the appropriate job control language for the hardware and operating system being used.

## USER ENVIRONMENT

It must be understood that the WSSM is not a program, but rather is a set of related programs and data files. As such, it is somewhat more complex in terms of installation than a single program or model with well-defined input and output. The purpose of the WSSM is to provide support technology to research, analysis, and planning efforts for water supply systems, and, as such, it is an open-ended tool, that can be utilized and extended in a number of different ways. The power of the WSSM can best be realized if it is treated not simply as a set of programs with fixed capabilities, but rather as a flexible set of tools for management, analysis, and display of network-oriented data. This requires, in general, that an individual become "involved" with, and knowledgeable about the details of the technology, in order to have the capacity to create new modules that respond to specific analysis needs. At the same time, however, the existing capabilities, and utility programs of the WSSM, do provide extensive and flexible manipulation capabilities, and can be used, without further extension, to simplify and organize a variety of water distribution analysis tasks.

## HARDWARE ENVIRONMENT

The WSSM can operate on medium to large computers having random access file capabilities. The data base files are organized as direct access files. File size is determined by the number of hydraulic elements (links and nodes) to be represented in a given data base. Program size for many of the programs is independent of file size, and is relatively small (under 64k bytes). Many of the analysis programs, however, use larger amounts of memory to accomodate

the relatively large network data bases (approximately 800 links) needed to provide either detailed or large-area representation of a water supply distribution system. If a smaller network is being analyzed, then these programs can be modified to reduce memory requirements.

As a practical matter, some means of providing graphical display is required for WSSM - typically, a pen plotter. Graphical display is indispensable to the checking of the data base, and is perhaps the most important tool available within the WSSM for synthesizing the results of complex analyses. The basic display program incorporated in WSSM, program NETPLOT, uses the de-facto industry standard Calcomp subroutine calling sequence for its graphic display calls.

The WSSM can be operated in either a batch or interactive mode - some of the programs presented here have interactive data specification, but could also be run in batch mode.

## HYDRAULIC NETWORK ANALYSIS MODEL

The WSSM incorporates an hydraulic network analysis model, that determines flow and velocity in links and pressure at nodes, given the physical and topological description of the network, and the demands at nodes. Many programs performing this function are available, using a variety of technologies. WSSM incorportates the WATSIM model, originally developed by Science Applications Inc. for the Office of Water Research and Technology.[3] This original implementation was for a UNIVAC computer, and had certain machine-specific features. A version of WATSIM, containing some corrections and enhancements, was obtained from the Denver Water Board, and used in the UNIVAC hardware development phase of WSSM. The MTS implementation uses an IBM-oriented version of WATSIM, obtained from the Public Service Company of Colorado. Minor modifications were made to this code to accomplish the necessary interface to the

WSSM data base, but the program functions essentially as described in the original WATSIM documentation, referenced above. In transporting WATSIM from one computer environment to another, the code was found to have certain minor quirks that required correction; accordingly, transport of WATSIM to other machines or operating systems may require adjustment.

If an analysis model other than WATSIM is desired, then it will be necessary to revise the two interface programs WATINT and WATWSSM to be consistent with the particular formats and requirements of the desired hydraulic analysis model. WATINT takes information from the data base and formats it to be consistent with WATSIM input requirements, and WATWSSM takes output from WATSIM and inserts it back into the data base.

Detailed documentation of the WATSIM model itself is not provided in this document, as it is available elsewhere.[3] Modifications made to WATSIM to accomplish the interface to WSSM data files are described later in this volume.

The SOLVER module, that solves simultaneous linear equations, is another of the analysis modules of WSSM that makes use of certain externally developed software. Subroutines that perform the sparse matrix solution technique were originally incorporated in the WATER program,[4] developed by Prof. M. Chandrashekar of the University of Waterloo, Ontario, and have been incorporated in the SOLVER program with his permission.

REPORT FORMAT

The remainder of this document consists of sections containing:

- user-level documentation of programs of WSSM
- programmer level documentation, including:
  - description of file structure and format
  - description of file access routines
  - documentation of subprogram and common blocks

- a description of the modifications to WATSIM
- discussion of transportability issues

Each of the major programs is listed as a major section below. Program source code listings will be made available through the National Technical Information Service. Where job control language is shown in example inputs, it is that used by the Michigan Terminal System (MTS) operating system of the University of Michigan Computer Center.

ERCHECK PROGRAM

PURPOSE: ERCHECK scans the node and link files searching for possible errors in the geographic and topologic data. It also produces a list of network ends, accompanied by the node type that is associated with that end.

OPTIONS: None

RESTRICTIONS: None

OVERVIEW: ERCHECK provides the user with information concerning the validity of topologic and geographic information via a list of error messages output to the printer. ERCHECK does not make any attempt to correct the data stored, only to point out possible sources of trouble. ERCHECK produces three lists: (1) a list of isolated links with their node identification (2) a list of the system ends, (3) a list of possible topologic and geographic errors.

INPUT DESCRIPTION:

There is no card input to this program.

PRINTER OUTPUT:

An example output can be found in figure 1. Below is a list of typical messsages found in the error message list.

IN NODE REC XXXXX PIPE XXXXX LISTED IN CONNECTIVITY LIST DOES NOT EXIST I/O ISTAT IS X

IN NODE REC XXXXX PIPE XXXXX IS IN CONNECTIVITY LIST. HOWEVER XXXXX XXXXX ARE ITS TERMINATION NODE IDS

TERMINATOR ENDS OF PIPE ARE IDENTICAL XXXXX XXXXX

NO VALID PIPE RECORD ASSOCIATED WITH NODE XXXXX
XXXXX XXXXX PIPES ARE PARALLEL BETWEEN NODES XXXXX XXXXX
PIPE XXXXX WITH NODE XXXXX NODE RECORD DOES NOT EXIST I/O ISTAT CODE IS X

The cause of each error message is explained within the message itself.

ABNORMAL PROGRAM TERMINATION:

None.

```
 TOPOLOGIC & GEOGRAPHIC ERROR SEARCH FOR NEW VIENNA FILES
  LARGEST USED NODE ID IS   50          LARGEST USED PIPE ID IS   57
  ERROR MESSAGE GENERATION ON 820323 AT 105958
     2    3 PIPES ARE PARALLEL BETWEEN NODES     2    3
     2    3 PIPES ARE PARALLEL BETWEEN NODES     3    2
 IN NODE REC    6 PIPE    6 LISTED IN CONNECTIVITY LIST DOES NOT EXIST I/O STAT IS 2
 ERR MESSAGE COMPLETED
03-23-8210:59:58
 NETWORK ENDS
 NODE     1      TYPE ONE      IN ZONE      4.
 NODE     4      TYPE ONE      IN ZONE      4.
 NODE     7      TYPE ONE      IN ZONE      4.
 NODE    11      TYPE ONE      IN ZONE      5.
 NODE    14      TYPE ONE      IN ZONE      4.
 NODE    31      TYPE ONE      IN ZONE      2.
 NODE    23      TYPE ONE      IN ZONE      6.
 NODE    24      TYPE ONE      IN ZONE      6.
 NODE    26      TYPE ONE      IN ZONE      7.
 NODE    33      TYPE TWO      IN ZONE      2.
 NODE    36      TYPE ONE      IN ZONE      8.
 NODE    50      TYPE  FOUR    IN ZONE      1.
 NODE    48      TYPE ONE      IN ZONE      1.
 NODE    49      TYPE ONE      IN ZONE      1.
```

**Figure 1: ERCHECK Output.**

ESTBLINK PROGRAM

PURPOSE: Establish the link file based on card image input describing link connectivity and physical characteristics.

OPTIONS: None

RESTRICTIONS: The node file establishment (ESTBNODE) program must have been executed prior to the execution of ESTBLINK. This module cannot be run twice against the same files with the same data without first re-executing the node establishment program.

OVERVIEW: ESTBLINK inserts the topological data into the WSSM files. It is the second program in the two program establishment procedure. The input cards describe how each link is to be connected into the system of nodes that were established by ESTBNODE. Geographic information from the node records as well as the attributes from the input card are stored into each logical link record. A record is kept of the largest identifier used. At the completion of the program, this value is stored into the link file's header record.

INPUT CARD DESCRIPTION:

The input deck consists of link definition cards accompanied by their characteristics. One card is required for each pipe. The cards do not have to appear in any specific order, nor do the pipes have to be sequentially numbered. The input deck end card has a '9999' in columns 1/4.

| FIELD | COLUMN | FORMAT | DESCRIPTION AND COMMENTS |
|---|---|---|---|
| 1 | 1/4 | I4 | pipe numeric identifier |
| 2 | 5/6 | I2 | pipe type (user specified) |
| 3 | 7/10 | I4 | pressure zone identifier |
| 4 | 11/15 | I5 | 1st connecting node id |
| 5 | 16/20 | I5 | 2nd connecting node id |
| 6 | 21/26 | I6 | date installed |
| 7 | 27/28 | I2 | material code (user specified) |
| 8 | 29/32 | I4 | secondary zone attribute |
| 9 | 33/40 | F8.4 | frictional coefficient (c-factor) |
| 10 | 41/50 | F10.0 | length (feet) |
| 11 | 51/55 | F5.0 | diameter (inches) |

| | | | |
|---|---|---|---|
| 12 | 56/60 | F5.0 | user specified factor (stored in pos. 16) |
| 13 | 61/65 | F5.0 | user specified factor (stored in pos. 17) |
| 14 | 66/70 | F5.0 | user specified factor (stored in pos. 18) |
| 15 | 71/75 | F5.0 | user specified factor (stored in pos. 19) |

A sample listing of an input deck for ESTBLINK is provided in Figure 2. As illustrated, not every attribute field needs to be filled for the establishment of the file. These attribute fields are provided for the user who has this information prepared for insertion at the outset of the file establishment process.

PRINTER OUTPUT:

A sample output from ESTBLINK is presented in Figure 3. Below are sample messages which will cause the program to reject the information received on a particular link definition card. The cause of the rejection is contained within the message itself.

```
RECORD NO XXXX MAY ALREADY EXIST APIPE(1) = XXXX.

NODE FILE ACCESS PROBLEMS
LINK XXXX WITH NODE XXXX HAS ISTAT XX

ERR IN WRITING TO THE LINK FILE
LINK XXXX WITH ISTAT X
```

ABNORMAL PROGRAM TERMINATION:

If a valid end card is not provided, then the I/O buffer will not be written back to the files, nor will the header records be updated. The user must determine the last valid record that was stored in the file, then rerun the program with the remainder of the card deck and a valid end card. If the user is unsure of the last record stored, the largest ID numbered card should be placed at the head of the input deck.

```
 1 1        2    155     2        130.     1040.   4.
 2 1        2    357     2        130.      220.   4.
 3 1        3    457     2        130.      120.   4.
 4 1        4    557     2        130.      550.   4.
 5 1        5    657     2        130.      180.   4.
 6 1        6    757     2        130.      550.   4.
 7 1        7    357     2        130.      120.   4.
 8 1        8    2       2        130.      360.   4.
 9 1        8    961     2        130.      330.   4.
10 1        9    1061    2        130.      700.   4.
11 1        10   1161    2        130.      720.   4.
12 1        12    855    1         80.     1130.   4.
13 1        16   12      1         80.      480.   4.
14 1        13   12      1         80.      460.   4.
15 1        15   16      1         80.      660.   4.
16 1        13   15      1         80.      530.   4.
17 1        13   14      1         80.      700.   4.
18 1        15   18      1         80.      330.   4.
19 1        17   1561    1        100.      720.   6.
20 1        17   13      1        100.      740.   4.
21 1        30   17      1        100.      350.   8.
9999
```

**Figure 2: ESTBLINK Input Deck (partial).**

```
 BEGIN LINK FILE ESTABLISHMENT     ON  820323  AT  105818
NODE FILE ACCESS PROBLEMS
 LINK      6 WITH NODE    67 HAS ISTAT     2 IBLOCK     1020
PROCESSING COMPLETE MAX. LINK NUMBER IS    57
 CARDS CORRECTLY USED     56 NO. OF REJECTED IS      1
03-23-8210:58:18
 ECHO        1     1     0     0     2     1     4.0000     1040.  550000  130.  0.  0.  0.
 ECHO        2     1     0     0     2     3     4.0000      220.  570000  130.  0.  0.  0.
 ECHO        3     1     0     0     2     3     4.0000      120.  570000  130.  0.  0.  0.
 ECHO        4     1     0     0     4     5     4.0000      550.  570000  130.  0.  0.  0.
 ECHO        5     1     0     0     5     6     4.0000      180.  570000  130.  0.  0.  0.
```

**Figure 3: ESTBLINK Sample Output (partial).**

ESTBNODE PROGRAM

PURPOSE: ESTBNODE creates a WSSM node file.

OPTIONS: None

RESTRICTIONS: A maximum of 790 nodes are permitted in the current version.

OVERVIEW: ESTBNODE is the first of a two step file establishment process for creating the WSSM files. The input card deck consists of node definition cards which contain the geographical location of the node, and the hydraulic element type associated with the node. ESTBNODE sets all other fields in the node record to the value "-9999".

INPUT CARD DESCRIPTION:

The input deck consists of a header card followed by node definition cards accompanied by their characteristics. One card is required for each node. The node cards do not have to appear in any particular order, nor do the nodes need to be sequentially numbered. The terminating card for this card deck contains the '9999' in card columns 1/4.

The header card specifies the maximum number of physical records to be made available in the node and link files. The number of physical records needed for a file is equal to the number of elements divided by ten, plus one.

HEADER CARD

| FIELD | COLUMN | FORMAT | DESCRIPTION AND COMMENTS |
|---|---|---|---|
| 1 | 1/5 | I5 | number of physical records for node file |
| 2 | 6/10 | I5 | number of physical records for link file |

NODE DESCRIPTION CARDS

| | | | |
|---|---|---|---|
| 1 | 1/4 | I4 | numeric node identifier |
| 2 | 5/6 | I2 | node type |
| 3 | 7/10 | F4.0 | elevation (in feet) |
| 4 | 11/20 | F10.0 | easting (feet) |
| 5 | 21/30 | F10.0 | northing (feet) |
| 6 | 31/34 | I4 | service zone |
| 7 | 35/37 | I3 | secondary zone |
| 8 | 38/47 | F10.4 | inflow or demand in cfs |

| | | | |
|---|---|---|---|
| 9 | 48/54 | F7.2 | node specific data for modeling purposes, (i.e. pump attributes, tank capacity, etc) or user specified attributes. These data items are stored in fields 24 through 27, respectively. |
| 10 | 55/61 | F7.2 | |
| 11 | 62/68 | F7.2 | |
| 12 | 69/75 | F7.2 | |

The geographic data is expected in state plane coordinates. Elevations are assumed to be to surface level except where the water system component is known to be significantly above or below ground. An example data input is shown in Figure 4.

PRINTER OUTPUT:

Example ouptut of ESTBNODE is shown in Figure 5. Below are sample error messages which will cause the program to reject the information received on a particular node definition card. The cause of the rejection is explained in the message itself.

NODE XXXXX PREVIOUSLY DEFINED CARD IGNORED

ERR IN WRITE XXXXXX. XXXX XX

ABNORMAL PROGRAM TERMINATION:

If the program terminates other than with a stop code of 0000, then a valid end card was not provided at the end of the card deck. The I/O buffer was not written to the files, nor was the header record filled. The program must be rerun in its entirety, after the files have been deleted.

```
10   10
 1 11137        80       1860   4        -0.086
 2 11125      1140       1840   5        -0.516
 3 11122      1140       2060   4        -0.086
 4 11124      1050       2150   4        -0.172
 5 11122      1100       2700   4        -0.344
 6 11125      1280       2700   4        -0.516
 7 11118      1230       2160   4        -0.086
 8 11123      1500       1840   4        -0.602
 9 11121      1500       1500   4        -0.516
10 11132       800       1520   5        -0.774
11 11136        80       1540   5        -0.516
12 11112      2640       1820   4        -0.602
13 11118      3100       1810   4        -1.460
14 11110      3000       2500   4        -0.688
15 11119      3300       1320   5        -2.971
16 11115      2640       1330   5        -0.258
```

**Figure 4: ESTBNODE Input (partial).**

```
BEGIN FILE ESTABLISHMENT PROCESS FOR WSSM  NODE FILE TO BE CREATED,LINK FILE TO BE ZEROED
 BEGUN ON 820323 AT 105551
 NODE FILE HAS    11 PHYSICAL RECORDS ROOM FOR    100 LOGICAL NODE RECORDS
LINK FILE HAS    11 PHYSICAL  RECORDS ROOM FOR    100 LOGICAL LINK RECORDS
 BAD INPUT CARD        3    1     1122.     1140.     2060.    4    0       -0.       0.       0.
PROCESSING COMPLETE FOR NODE FILE ESTABLISHMENT MAX NODE RECORD IS   50NUMBER OF CARDS PROCESSED IS   50
  NUMBER OF CARDS REJECTED IS     1
+++++ BOUNDING RECTANGLE ++++++++
            N      5690.
  E        80.          7760. E
            N       200.
+++++ BOUNDING RECTANGLE ++++++++
03-23-8210:55:51
  ECHO     1  11137.       80.      1860.    4    0   -0.    0.       0.       0.       0.
  ECHO     2  11125.     1140.      1840.    5    0   -1.    0.       0.       0.       0.
  ECHO     3  11122.     1140.      2060.    4    0   -0.    0.       0.       0.       0.
  ECHO     4  11124.     1050.      2150.    4    0   -0.    0.       0.       0.       0.
```

**Figure 5: ESTBNODE Sample Output (partial).**

FIXL PROGRAM

PURPOSE: FIXL will add, delete or redefine a link record.

OPTIONS: None

RESTRICTIONS: None

OVERVIEW: With the use of FIXL, the topological structure of a WSSM data file can be modified. Topological errors introduced at the time the files were established can be corrected using FIXL. This module, with FIXN, can be used to expand the existing data base by the addition of new nodes and links to the system.

INPUT CARD DESCRIPTION:

The input deck to FIXL consists of a deletion deck and an "add/redefine" deck. The add/redefine deck must be followed by a valid end card which has a '99999' in card columns 1/5. The deletion deck need not be present if only "adds" and "redefines" are to be made; similarly the add/redefine deck need not be present if only deletions are to be made. The end card must always be present.

DELETION CARD DECK:

If this deck is present, it must be the first input deck to be processed.

| FIELD | COLUMN | FORMAT | DESCRIPTION AND COMMENTS |
|---|---|---|---|
| 1 | 1/5 | I5 | the link numeric ID of the link record to be deleted, preceded by a (-) minus sign. |

ADD/REDEFINE CARD DECK:

| FIELD | COLUMN | FORMAT | DESCRIPTION AND COMMENTS |
|---|---|---|---|
| 1 | 1/5 | I5 | the numeric ID of the link record to be created or redefined |
| 2 | 6/10 | I5 | ID of upstream node for link |
| 3 | 11/15 | I5 | ID of downstream node for link |
| 4 | 21/30 | F10.0 | length (feet) |
| 5 | 31/40 | F10.0 | diameter (inches) |

A sample input deck for FIXL is presented in Figure 6.

PRINTER OUTPUT:

An example output can be found in Figure 7. Below is a list of the error messages that may be generated by FIXL when it is unable to carry out the request of the user. These messages explain the reason the program terminated the editing sequence requested on the input card.

LINK REC# XXXX DELETION IMPEDED BY ISTAT= X

NODE REC# XXXX TO BE UPDATED, CANNOT BE ACCESSED
ANODE(1)= XXXX. AND ISTAT= X

NO ROOM IN NODE RECORD XXXX TO STORE XXXX

XXXX PIPE NOT FOUND IN XXXX NODE LIST OF CONNECTED PIPES

OUTPUT ERR IN WRITE ATTEMPT TO NODE # XXXX ANODE(1)= XXXX. AND ISTAT= X

```
$RUN W.FIXL.0 6=-OUTPUT 10=-TIME 11=NEWV3.DA.NODE 12=NEWV3.DA.PIPE
  -13
   13   21    8           4.     1600.
99999
$ENDFILE
```

**Figure 6: FIXL Sample Input.**



```
BEGIN TOPOLOGICAL EDITING VIA PIPE FILE ON   820303 AT   121911
INPUT CARD  -13    0  0   CORRECTIONS FOR DIAMETER   0. AND LENGTH         0.
 13 PIPE CONNECTING NODE    21 TO NODE     8 IS  TO BE DELETED !!!!!
  21 :PIPE LIST OLD:    26   27   13    0  0  0  0  0  0  0 NEW:    26   27    0  0  0  0  0  0  0  0
   8 :PIPE LIST OLD:     8    9   13    0  0  0  0  0  0  0 NEW:     8    9    0  0  0  0  0  0  0  0
PIPE RECORD    13 IS DELETED FROM FILE
INPUT CARD   13   21    8     CORRECTIONS FOR DIAMETER   4. AND LENGTH      1600.
 13 PIPE  *****  NEW RECORD ***** CONNECTS-NODE    21 TO NODE     8
  21 :PIPE LIST OLD:    26   27    0  0  0  0  0  0  0  0 NEW:    26   27   13    0  0  0  0  0  0  0
   8 :PIPE LIST OLD:     8    9    0  0  0  0  0  0  0  0 NEW:     8    9   13    0  0  0  0  0  0  0
END EDIT OF PIPE 13 (X,Y,Z)N1: 2300.  570. 1118. (X,Y,Z)N2: 1500. 1840. 1123. DIAMETER=4.LENGTH=1600.
INPUT CARD99999    0  0   CORRECTIONS FOR DIAMETER   0. AND LENGTH         0.
END TOPOLOGICAL EDITING
```

**Figure 7: FIXL Sample Output.**

FIXN PROGRAM

PURPOSE: FIXN is an editing module which can add, or delete, or change a node, or change node coordinates of an existing node.

OPTIONS: None

RESTRICTIONS: None

OVERVIEW: FIXN is used to make corrections to the topologic and geographic data stored within the WSSM node file. FIXN also makes the necessary geographic corrections to the link file so that there is consistency between the data stored within the node file and the link file. When a node record is removed from the file, all links which have that node as a downstream node will also be deleted. If FIXN and FIXL are to be used in sequence to add additional nodes and pipes to the system, FIXN must be executed first.

INPUT CARD DESCRIPTION:

The input deck to FIXN consists of a deletion deck and/or an "add/relocate" deck. A valid end card with a '99999' in columns 1/5 must be the last input card. The deletion deck need not be present if no nodal records are to be deleted; similarly, an add/relocate deck need not be present if only deletions are to be made.

DELETION CARD DECK

If this deck is to be present, it must be the first input deck to be processed.

| FIELD | FORMAT | COLUMN | DESCRIPTION AND COMMENTS |
|---|---|---|---|
| 1 | I5 | 1/5 | the node numeric ID of the record to be deleted, preceded by a (-) minus sign. |

ADD/RELOCATE CARD DECK

| FIELD | FORMAT | COLUMN | DESCRIPTION AND COMMENTS |
|---|---|---|---|
| 1 | I5 | I5 | the node numeric ID of the record to be added or relocated |
| 2 | 11/20 | F10.0 | easting (feet) |
| 3 | 21/30 | F10.0 | northing (feet) |
| 4 | 31/40 | F10.0 | elevation (feet) |

A sample FIXN input deck is illustrated in Figure 8.

PRINTER OUTPUT:

Below is a list of the error messages that may be generated by FIXN when it is unable to carry out the request of the user. These messages explain the reason the program terminated the editing sequence requested on the input card.

D.A. WRITE ERR IN NODE FILE: (INODE,ANODE(1),ISTAT)= XXX XXXX. X

D.A. READ ERR OF NODE FILES:(INODE,ANODE(1),ISTAT)= XXX XXXX. X

D.A. READ ERR OF PIPE FILE:(IPIPE,APIPE(1),ISTAT)= XXX. XXXX X

NODE XXX HAS PIPE XXX AS BEING CONNECTED HOWEVER ITS ENDING NODES ARE
XXX XXX

NODE TO BE DELETED IS NOT IN FILE ISTAT= X

D.A. WRITE ERR IN PIPE FILE:(IPIPE,APIPE(1),ISTAT) XXX XXXX. X

NODE NOT DELETED ISTAT= XXX

WHILE DELETING NODE XXX PIPE XXX WAS ACCESSED WITH NODES XXX XXX
PIPE NOT DELETED

PIPE NOT DELETED ISTAT= X

A sample output showing a "normal" termination of FIXN is shown in Figure 9.

```
$RUN W.FIXN.U+W.IUMODULE.U+W.BRWRIT.U+W.DATIME.U 6=-OUTPUT 11=NEWV3.DA.NODE 12=NEWV3.DA.PIPE 10=-TIME
  -46
   12                          1112.
99999
$ENDFILE
```

**Figure 8: FIXN Sample Input.**

```
 BEGIN COORDINATE EDITING OF NODE FILE ON  820401 AT  113429
 START EDIT FROM INPUT CARD  -46        0.        0.        0.
NODE:    46 ZEROED IN NODE FILE
LINK:    50 DELETED FROM LINK FILE
LINK:    52 DELETED FROM LINK FILE
 START EDIT FROM INPUT CARD   12        0.        0.     1112.
 NODE   12 OLD COORD:      2640.     1820.     1080. CORRECTIONS:        0.        0.     1112.
    LINKS CONNECTED TO NODE   12 ARE      14.      0.  0.  0.  0.  0.  0.  0.  0.  0.
    UPDATE PIPE    14 FROM NODE    12 OLD COORD: 2640.  1820.  1080. NEW COORD:  2640.  1820.  1112.
 END EDIT OF NODE    12
 START EDIT FROM INPUT CARD99999        0.        0.        0.
  NODE FILE EDITING COMPLETED, FILES CLOSED
```

**Figure 9: FIXN Sample Output.**

FLEXI PROGRAM

PURPOSE: FLEXI provides the user with a simple method of performing basic algebraic manipulations on data in the link and node files, and storing the result back into the data file.

OPTIONS: Constant values can be assigned to parameters in the equation.

RESTRICTIONS: Data cannot be inserted into restricted columns, or into a column used for calculation.

OVERVIEW: It is frequently desirable to perform such functions as comparisons between two different pressures (based on two different scenarios) stored in the node file, or to convert data from a unit basis to a total flow basis. While the WSSM does not have a complete algebraic data management capability, program FLEXI provides some capacity in this regard. FLEXI will perform calculations of the form:

$$X = (A \pm B)*C/D$$

where A, B, C, and D are columns in the node or link file, and X is an output column into which the resultant calculation is to be placed for each node or link record. If desired, B, C, or D may be set to constant values rather than to columns in the file. Parameter A, however, must be a valid column in the file. The calculations are carried out for all records in the file.

PROGRAM INPUT:

The type of calculation to be performed by FLEXI is determined by three input cards. The first card specifies whether to treat the equation form as A+B or as A-B, selects the node or link file, and defines the output storage column. The second card specifies the columns to be used as input for parameters A, B, C, and D. If the field corresponding to the parameter is left blank, then the parameter is not used in the calculation, or a constant parameter, as specified on the third card, is used. The third card specifies whether or not constant parameters are to be used, and defines the values of those parameters. Card image input is read by FLEXI on logical unit 5. Detailed information on the usage of each input card is given below:

| Card | Field | Format | Card Column | Description |
|---|---|---|---|---|
| 1 | 1 | I1 | 1 | Addition mode. If field =0, then use A+B, if greater than 0, then calculate as A-B. |
| 1 | 2 | A4 | 2/5 | File Selection - "NODE" or "LINK" storage column, |
| 1 | 3 | I5 | 6/10 | must be greater than 25, and less than or equal to maximum permitted for file type. |

| | | | | |
|---|---|---|---|---|
| 2 | 1 | I5 | 1/5 | Valid column location for variable A (this is required, and cannot be left blank). |
| 2 | 2 | I5 | 6/10 | Column location for variable B (if blank, then either not used or overridden by value on card 3). |
| 2 | 3 | I5 | 11/15 | Column location for variable C. |
| 2 | 4 | I5 | 16/20 | Column location for variable D. |
| 3 | 1 | F10.0 | 1/10 | Constant parameter switch. If this field is less than 0, then there are no fixed parameters. If 0 or greater, then fixed parameters are used. The value has no other significance. |
| 3 | 2 | F10.0 | 11/20 | Constant value to be used in place of variable B. |
| 3 | 3 | F10.0 | 21/30 | Constant value to be used in place of variable C. |
| 4 | 4 | F10.0 | 31/40 | Constant value to be used in place of variable D. |

PROGRAM OUTPUT:

Program output is presented on logical unit 6, echoing the input, and on unit 10, providing the calculated value for each record. An example of program input and output is shown in Figures 10 and 11.

```
$RUN W.FLEXI.U 6=-OUTPUT 10=-TIME 11=NEWV3.DA.NODE 12=NEWV3.DA.PIPE
ONODE    38
   39
                                1.314
$ENDFILE
```

**Figure 10: FLEXI Input (with Runner).**

```
DATE:  820414                          NEW VIENNA WATER
TIME:  145358          CALCULATION NODE#  0 FOR NODE WITH VALUE STORED IN COLUMN   38
CALCULATIONS COMPLETE FOR      NEW VIENNA WATER
```

**Figure 11a: FLEXI Output (logical unit 6).**

```
04-14-8214:53:58
  LOGICAL RECORD     VALUE
     NUMBER
         1           248.19
         2           196.11
         3           109.47
         4           125.34
         5           165.25
         6           174.57
         7           117.37
         8            94.62
         9           104.78
        10           133.89
        11           177.37
        12            92.66
        13            59.24
        14            86.25
        15            55.17
        16            94.15
        17            48.96
```

**Figure 11b: FLEXI Output (logical unit 10). (partial)**

INSELK PROGRAM

PURPOSE: INSELK inserts attribute data into specified locations in the link file.

OPTIONS: None

RESTRICTIONS: Link record fields 1, and 5 through 12 are protected from the insertion process ("write protected").

OVERVIEW: INSELK inserts data directly into the file. The program checks the columns for insertion against a list of write protected columns. If any of the requested columns of a single card is flagged as write protected, the entire insertion card is rejected. No other check is made upon the data being stored, or the columns in which it is to be stored.

INPUT CARD DESCRIPTION:

The input deck consists of a series of insertion cards. One card is required for each record to be updated. Up to six values may be inserted into a single record for each insertion card. If more than six attributes are to be updated for a single link, then more than one insertion card will be required. The last card in the insertion deck should be the end card which contains a '9999' in columns 1/4 of the card.

| FIELD | COLUMN | FORMAT | DESCRIPTION AND COMMENTS |
|---|---|---|---|
| 1 | 1/4 | I4 | record ID to be updated |
| 2 | 9/10 | I2 | column to store first insertion |
| 3 | 11/18 | F8.0 | first insertion value |
| 4 | 19/20 | I2 | column for 2nd insertion |
| 5 | 21/28 | F8.0 | 2nd value to be inserted |
| 6 | 29/30 | I2 | column for third insertion |
| 7 | 31/38 | F8.0 | 3rd value to be inserted |
| 8 | 39/40 | I2 | column for fourth insertion |
| 9 | 41/48 | F8.0 | 4th value to be inserted |
| 10 | 49/50 | I2 | column for fifth insertion |
| 11 | 51/58 | F8.0 | 5th value to be inserted |
| 12 | 59/60 | I2 | column for sixth insertion |
| 13 | 61/68 | F8.0 | 6th value to be inserted |

PRINTER OUTPUT:

Below is a list of the error messages that may be generated by INSELK when an insertion is aborted by the program.

INSERTION TERMINATED FOR LINK RECORD XXXXX COLUMNS WRITE PROTECTED.
ONE OR MORE IS IN ERR XX XX XX XX XX XX

ERR IN READ ACCESS ON NODE FILE: ANODE(1)= XXXX. For Link Record #XXX IO ISTAT CODE = X

Sample input and output are shown in Figures 12 and 13.

```
$RUN W.INSELK.O 6=-OUTPUT 10=-TIME 11=NEWV3.DA.NODE 12=NEWV3.DA.PIPE
  15    21 130.   26 375.70 27 5000.  28 43.40  29 1981.  31 332.30
  15    36 1.
9999
$ENDFILE
```

**Figure 12: INSELK Sample Input (with Runner).**

```
 BEGIN INSERTION TO PIPE FILE ON   820303 AT   140432  PROGRAM UPDATE CODE IS 3040
   15 PIPE RECORD
 21COLUMN VALUE        130. REPLACES          0. 26COLUMN VALUE        376. REPLACES          0.
 27COLUMN VALUE       5000. REPLACES          0. 28COLUMN VALUE         43. REPLACES          0.
 29COLUMN VALUE       1981. REPLACES          0. 31COLUMN VALUE        332. REPLACES          0.
   13 PIPE RECORD
 36COLUMN VALUE          1. REPLACES          0.
FILES CLOSED HEADER UPDATED ON   820303 AT  140432 CODE= 3040
```

**Figure 13a: INSELK Sample Output (output on logical unit 6).**

```
03-03-8214:04:32
                                 COL. VALUE   COL. VALUE   COL. VALUE   COL. VALUE   COL. VALUE   COL. VALUE
  15 CORRECTIONS;(COLUMN,VALUE) 21      130. 26      376. 27     5000. 28       43. 29     1981. 31      332.
  15 CORRECTIONS;(COLUMN,VALUE) 36        1. 0         0. 0        0. 0        0. 0        0. 0        0.
```

**Figure 13b: INSELK Sample Output (output on logical unit 10).**

INSEND PROGRAM

PURPOSE:

To insert attribute data into specified locations of the node file.

OPTIONS: None

RESTRICTIONS:

Fields 1, and 5 through 17 are protected from the insertion process.

OVERVIEW: INSEND inserts data directly into the node file under control of input cards. The program checks the columns on the input card against a list of write-protected columns. If any of the requested columns on a single card is flagged as write-protected, the entire insertion card is rejected. No other check is made upon the data being stored, or the columns in which it is to be stored.

INPUT CARD DESCRIPTION:

The input deck consists of a series of insertion cards. One card is required for each record to be updated. Up to six values may be inserted into a single record for each insertion card. If more than six attributes are to be updated for a single node, then more than one insertion card will be required. The last card in the insertion deck should be the end card which contains a '9999' in columns 1/4 of the card.

| FIELD | COLUMN | FORMAT | DESCRIPTION AND COMMENTS |
|---|---|---|---|
| 1 | 1/4 | I4 | record ID to be updated |
| 2 | 9/10 | I2 | column to store first insertion value |
| 3 | 11/18 | F8.0 | first insertion value |
| 4 | 19/20 | I2 | column for 2nd insertion |
| 5 | 21/28 | F8.0 | 2nd value to be inserted |
| 6 | 29/30 | I2 | column for third insertion |
| 7 | 31/38 | F8.0 | 3rd value to be inserted |
| 8 | 39/40 | I2 | column for fourth insertion |
| 9 | 41/48 | F8.0 | 4th value to be inserted |
| 10 | 49/50 | I2 | column for fifth insertion |
| 11 | 51/58 | F8.0 | 5th value to be inserted |
| 12 | 59/60 | I2 | column for sixth insertion |
| 13 | 61/68 | F8.0 | 6th value to be inserted |

PRINTER OUTPUT:

Below is a list of the error messages that may be generated by INSEND when an insertion is aborted by the program. These messages should explain the reason the program terminated the insertion requested by the particular insertion card.

```
INSERTION TERMINATED FOR NODE RECORD XXXXX COLUMNS WRITE PROTECTED.
ONE OR MORE IS IN ERR  XX  XX  XX  XX  XX  XX

ERR IS READ ACCESS ON NODE FILE: ANODE(1)= XXXX.  FOR NODE RECORD
# XXXX IO ISTAT CODE = X

ERR IN WRITE ACCESS ON LINK FILE: ANODE(1) = XXXX.  FOR LINK
RECORD # XXX IO ISTAT CODE = X
```

Sample input and output for program INSEND are shown in Figures 14 and 15.

```
$RUN W.INSEND.0 6=-OUTPUT 10=-TIME 11=NEWV2.DA.NODE 12=NEWV2.DA.PIPE
  33     32  1240. 33  1245.
  44     32  1240. 33  1245.
9999
$ENDFILE
```

**Figure 14: INSEND Sample Input (with Runner).**

Reproduced from best available copy.

```
 BEGIN INSERTION TO NODE FILE ON   820203 AT   163357  PROGRAM UPDATE CODE IS 3020
   33 NODE RECORD
 32COLUMN VALUE    1240. REPLACES        0. 33COLUMN VALUE    1245. REPLACES        0.

   44 NODE RECORD
 32COLUMN VALUE    1240. REPLACES        0. 33COLUMN VALUE    1245. REPLACES        0.

FILES CLOSED HEADER UPDATED ON   820203 AT  163357 CODE= 3020
```

**Figure 15: INSEND Sample Output (logical unit 6).**

LCOPYATT PROGRAM

PURPOSE: To copy one column in the link file into another column of the link file, or to set all of the values in a column to a specific 'background' value.

OPTIONS: Either a column may be copied, or set to a given value.

RESTRICTIONS: The column in which the copied data is inserted must be in the range from 25 to 40.

OVERVIEW: LCOPYATT is a general-purpose utility that simplifies certain operation of data definition within WSSM. By allowing the user to copy data from one column to another, 'backup' columns can be used to save information. Further, tasks such as developing a scenario are simplified, particularly if it represents only a modest change from a previously-defined and stored scenario. Using LCOPYATT, the initial types can be set to a single value using the background feature, and program INSELK can then be run to change those link elements that have different types. Typically, the majority of pipe types will be simple pipes, and only a few will need to be changed. LCOPYATT operates on all records in the file. If copies or background values are to be applied to more than a single column, then multiple runs of LCOPYATT must be carried out. LCOPYATT is completely analogous in performance to program NCOPYATT which operates on the node file.

PROGRAM INPUT:

Input consists of a single card, containing two or three values.

| Field | Column | Format | Description |
|---|---|---|---|
| 1 | 1/2 | I2 | If positive, the source column to be copied. If less than zero, indicator that a background value operation, rather than a copy operation is to be carried out. |
| 2 | 3/5 | I3 | Destination column, into which copied background data will be placed for each record (must be between 25 and 40). |
| 3 | 6/15 | F10.0 | The background value to be inserted, if the first field of the input card is negative - ignore otherwise. |

PROGRAM OUTPUT:

Program output consists of a set of echo messages defining the run conditions as specified on the input card, and indicating successful or unsuccessful completion of the LCOPYATT program.

A sample input and output for program LCOPYATT are shown in Figures 16 and 17.

```
$RUN W.LCOPYATT.O 6=-OUTPUT 10=-TIME 11=NEWV3.DA.NODE 12=NEWV3.DA.PIPE
-1 39  100.
$ENDFILE
```

**Figure 16: LCOPYATT input, background mode.**



```
COPYATT FOR FILE      NEW VIENNA WATER        ON 820414 AT 150545 FILE"S LARGEST PIPE NUMBER IS    57
INPUT ECHO:  -1  39        100.
COLUMN    39 TO BE BACK VALUED BY         100.
COPYATT PROCESS COMPLETED FOR FILE      NEW VIENNA WATER        FOR    49 PIPES
COPYATT PROCESS COMPLETED FOR FILE 2
```

**Figure 17: LCOPYATT output.**

NAMEIN PROGRAM

PURPOSE: NAMEIN inserts an alphabetic name of the WSSM files into the node file header record.

OPTIONS: None

RESTRICTIONS: None

OVERVIEW: NAMEIN inserts a 28-character title into the node file header record, beginning at field 108 of the header.

INPUT CARD DESCRIPTION:

The name card has one field, in columns 1/28. It is read in format 28A1.

PRINTER OUTPUT:

A single message is printed at the completion of the name insertion process:

NAME XXXXXXXXXXXX HAS BEEN INSERTED INTO THE FILE.

NCOPYATT PROGRAM

PURPOSE: To copy one column in the node file into another column of the node file, or to set all of the values in a column to a specific 'background' value.

OPTIONS: A column may be copied from another column, or set to a given value.

RESTRICTIONS: The column in which the copied data is inserted must be in the range from 25 to 60.

OVERVIEW: NCOPYATT is a general-purpose utility that simplifies certain operation of data definition within WSSM. By allowing the user to copy data from one column to another, 'backup' columns can be used to save information. Further, tasks such as developing a scenario are simplified, particularly if it represents only a modest change from a previously-defined and stored scenario. Using NCOPYATT, the initial types can be set to a single value using the background feature, and program INSEND can then be run to change those nodal elements that have different types. Typically, the majority of nodal types will be simple demands, and only a few will need to be set to fixed head. NCOPYATT operates on all records in the file. If copies or background values are to be applied to more than a single column, then multiple runs of NCOPYATT must be carried out. NCOPYATT is completely analogous in performance to program LCOPYATT which operates on the link file.

PROGRAM INPUT:

Input consists of a single card, containing two or three values.

| Field | Column | Format | Description |
|---|---|---|---|
| 1 | 1/2 | I2 | If positive, the source column to be copied.<br>If less than zero, indicator that a background value operation, rather than a copy operation is to be carried out. |
| 2 | 3/5 | I3 | Destination column, into which copied background data will be placed for each record (must be between 25 and 60). |
| 3 | 6/15 | F10.0 | The background value to be inserted, if the first field of the input card is negative - ignored otherwise. |

PROGRAM OUTPUT:

Program output consists of a set of echo messages defining the run conditions as specified on the input card, and indicating successful or unsuccessful completion of the NCOPYATT program.

A sample input and output for program NCOPYATT are shown in Figures 18 and 19.

```
$RUN W.NCOPYATT.0 6=-OUTPUT 10=-TIME 11=NEWV3.DA.NODE 12=NEWV3.DA.PIPE
31 54
$ENDFILE
```

**Figure 18: NCOPYATT Sample Input.**



```
COPYATT FOR FILE      NEW VIENNA WATER        ON 820414 AT 150044 FILE"S LARGEST NODE NUMBER IS    50
INPUT ECHO:  31  54          0.
COPYATT PROCESS COMPLETED FOR FILE      NEW VIENNA WATER        FOR    47 NODES
COPYATT PROCESS COMPLETED FOR FILE
```

**Figure 19: NCOPYATT Sample Output.**

NETPLOT PROGRAM

PURPOSE:

NETPLOT creates pen plotter plots of various types, of the entire system or a portion thereof at any desired scale.

OPTIONS:

Variable scaling or automatic scaling to paper width
Variable paper size
Pipe representation by 1) line 2) arrow 3) not drawn
Grid overlay labeled in feet
Paper width of grid cell
Plot a box around plot
Plot a symbol (or not) at location of node
Plot up to three attribute values at each node or pipe
Size of numbers used in attribute plot
Override the node symbology with user specifications
Selective plotting of pipes and nodes

RESTRICTIONS:

NETPLOT uses "industry-standard" Calcomp-type graphic subroutines, which must be available at the WSSM installation.

OVERVIEW:

As many plots as are desired for a single system may be made with each run of the NETPLOT. The paper width specified on the input card represents the maximum width of paper that can be used by the plot. If the plot request cannot be adequately scaled (with the user specified scale) to that width, then the program will automatically break the plot up into windows. Conversely, if the plot will fit onto a smaller piece of paper than requested, the module will reduce the plot limits accordingly to prevent the border box from being drawn around largely empty space. The minimum paper width that the program will allow is 10 inches - the width of the title box that accompanies each plot window. Currently the program does not allow overlay plotting; to protect the integrity of the plot generated, the program moves the pen location 10 inches past the plot generated and re-origins it at that point. Many of the specific plot functions are separable so that a wide variety of plots can be generated by this module.

INPUT CARD DESCRIPTION:

NETPLOT reads the control cards on logical unit 5. The information contained on the control cards specifies the type of plot that is to be created. There are four control cards for each plot that is to be generated. A valid end card should follow the last control card to the last plot request to properly close the plot file and move the plot pen beyond the surface upon which the plots were generated.

CONTROL CARD 1

| FIELD | COLUMN | FORMAT | DESCRIPTION AND COMMENTS |
|---|---|---|---|
| 1 | 1/5 | F5.0 | paper width to be used, in inches (default=34) |
| 2 | 11/20 | F10.8 | scale chosen to generate plot (feet/feet) If set to 24000, one plot inch represents 24000 inches. If this field is left blank, the automatic scaling option is used, producing a single window plot of the file which completely uses the paper width specified in field 1. |
| 3 | 21/30 | F10.8 | fields 3 through 6 are used to specify a smaller window of the file to be plotted. Field 3 has the maximum easting limits, in user file coordinates typically feet. |
| 4 | 31/40 | F10.8 | minimum easting for the window |
| 5 | 41/50 | F10.8 | maximum northing for the window |
| 6 | 51/60 | F10.8 | minimum northing for the window |

CONTROL CARD 2

| FIELD | COLUMN | FORMAT | DESCRIPTION AND COMMENTS |
|---|---|---|---|
| 1 | 1/5 | I5 | =0 pipe not represented in plot<br>=1 pipe is represented by a line<br>< =1 pipe is represented by an arrow indicating flow direction. The value in Field 1 is the column in the link file in which the flow is stored. |
| 2 | 6/10 | I5 | =1 draw box around plot<br>≠1 do not draw box around plot |
| 3 | 11/15 | I5 | ≠0 draw a grid overlay<br>=0 do not draw grid overlay |
| 4 | 16/20 | I5 | this switch controls the link attribute and number plotting at a point midway between pipe ends<br>=0 no plot of number or attribute values<br>=1 plot number only not attributes<br>=2 plot attributes and number |

INPUT DESCRIPTION (Contd)
CONTROL CARD 2 (Contd)

| FIELD | COLUMN | FORMAT | DESCRIPTION AND COMMENTS |
|---|---|---|---|
| 5 | 21/25 | I5 | This switch controls the plotting of (a) node symbols (b) node numbers and (c) nodal attributes<br>=-3 plot symbol and attributes<br>=-2 plot symbol, attributes and number<br>=-1 plot symbol and number<br>=0 no plot of these items<br>=1 plot number<br>=2 plot number and attributes<br>=3 plot attributes only |
| 6 | 26/30 | I5 | size of numbers to be plotted in hundredths of an inch |
| 7 | 31/35 | I5 | inch width of grid overlay cell |
| 8 | 36/40 | I5 | column in which link attribute 1 is stored |
| 9 | 41/45 | I5 | column in which link attribute 2 is stored |
| 10 | 46/50 | I5 | column in which link attribute 3 is stored |
| 11 | 51/55 | I5 | column in which node attribute 1 is stored |
| 12 | 56/60 | I5 | column in which node attribute 2 is stored |
| 13 | 61/65 | I5 | column in which node attribute 2 is stored |
| 14 | 66/70 | I5 | column number for selective attribute plotting |
| 15 | 71/75 | F10.0 | value in selected column for which plotting is desired. |

CONTROL CARD 3

| FIELD | COLUMN | FORMAT | DESCRIPTION AND COMMENTS |
|---|---|---|---|
| 1/9 | 1-45 | 15 | If the user wishes to override the default node symbols he may specify those symbols desired by entering the standard "Calcomp" symbol number in the corresponding field position |

CONTROL CARD 4

| FIELD | COLUMN | FORMAT | DESCRIPTION AND COMMENTS |
|---|---|---|---|
| 1 | 1/32 | 8A4 | plot title to be written into the plot legend |

CONTROL CARD 5
The end card has a '99999' in columns 1/5

INPUT DESCRIPTION (Contd)

Sample inputs for two runs of NETPLOT are shown in Figures 20a and 20b. The input specification of Figure 20a requests a plot of the schematic of a data base, with node and link numbers, and default plot symbols for nodal type. The plot is to be automatically scaled to fit into an 11" high space, and links are to be represented as lines. Annotation is to be 14 hundredths of an inch high at the original scale. (The display examples shown in Figure 21 are not at original scale). The input example of Figure 20b requests a plot that is automatically scaled into a height of 18 inches, showing flow direction (links represented by arrows), with flow direction being determined by the flow quantities stored in column 34 of the link file, and head at each node, with head values being drawn from column 36 of the node file. Annotation is seven hundredths of an inch high in the original.

EXAMPLE OUTPUT

Printer output accompanying a plot consists of an "echo" of the control cards, descriptions of the run conditions, giving scale, method of representation of links, etc., error messages, and ending statements. Examples of the corresponding printer output for input examples of Figure 20a and 20b are shown in Figures 21a and 21b, and portions of the corresponding plots are shown in Figures 21c and 21d. It should be noted that these portions of the plots are not reproduced at the original scale, thus the annotation height does not correspond to that specified in the input. In addition, the original plots use color to separate certain of the features, allowing for easier interpretation.

```
11.
    1                1    -1    14

NEW VIENNA LINK NODE NETWORK
99999
```

**Figure 20a: NETPLOT Sample Input.**

```
  18
  34    0    0    0    3   0/                          36

SCENARIO # 13 HEADS
99999
```

Reproduced from best available copy.

**Figure 20b: NETPLOT Sample Input.**

```
                                NEW VIENNA LINK NODE NETWORK
 JPEN=  1;IBOX=  0;IGRID  0;LW=  1:NW= -1
 IZ= 14;ISKIP=  0:LPARM=  0  0  0;NPARM=  0  0  0;ICOL=  0;IVAL=          0
 JSYL=  0  0  0  0  0  0  0  0  0
 JSYL=  0  0  0  0  0  0  0
 AUTO SCALE SELECTED
  BOUNDING RECTANGLE IS :(X,Y),(MAX,MIN)    7760.0      80.0    5690.0     200.0
  SCALE IS  500.00000 NEW PEN RECTANGLE IS      15.4       0.0      11.0       0.0
 PLEN= :  11.NUML= :   5/NLOOP    1
  PARTIALLY PLOTTED PIPES THIS WINDOW IS    0
 WHOLE PIPES PLOTTED THIS WINDOW IS   56
     1 WINDOWS SUCCESSFULLY PLOTTED
```

**Figure 21a: NETPLOT Printer Output.**

```
                                SCENARIO # 13 HEADS
 JPEN= 34;IBOX=  0:IGRID  0;LW=  0;NW=  5
 IZ=  7;ISKIP=  0:LPARM=  0  0  0;NPARM= 36  0  0;ICOL=  0;IVAL=          0
 JSYL=  0  0  0  0  0  0  0  0  0
 JSYL=  0  0  0  0  0  0  0
 LINK REPRESENTED BY ARROWS
 AUTO SCALE SELECTED
  BOUNDING RECTANGLE IS :(X,Y),(MAX,MIN) 194/500.0 18/9000.0  5/9550.0  510000.0
  SCALE IS 3864.00000 NEW PEN RECTANGLE IS      17.7       0.0      18.0       0.0
 PLEN= :  18.NUML= :  200NLOOP    1
  PARTIALLY PLOTTED PIPES THIS WINDOW IS    0
 WHOLE PIPES PLOTTED THIS WINDOW IS  161
     1 WINDOWS SUCCESSFULLY PLOTTED
```

**Figure 21b: NETPLOT Printer Output.**

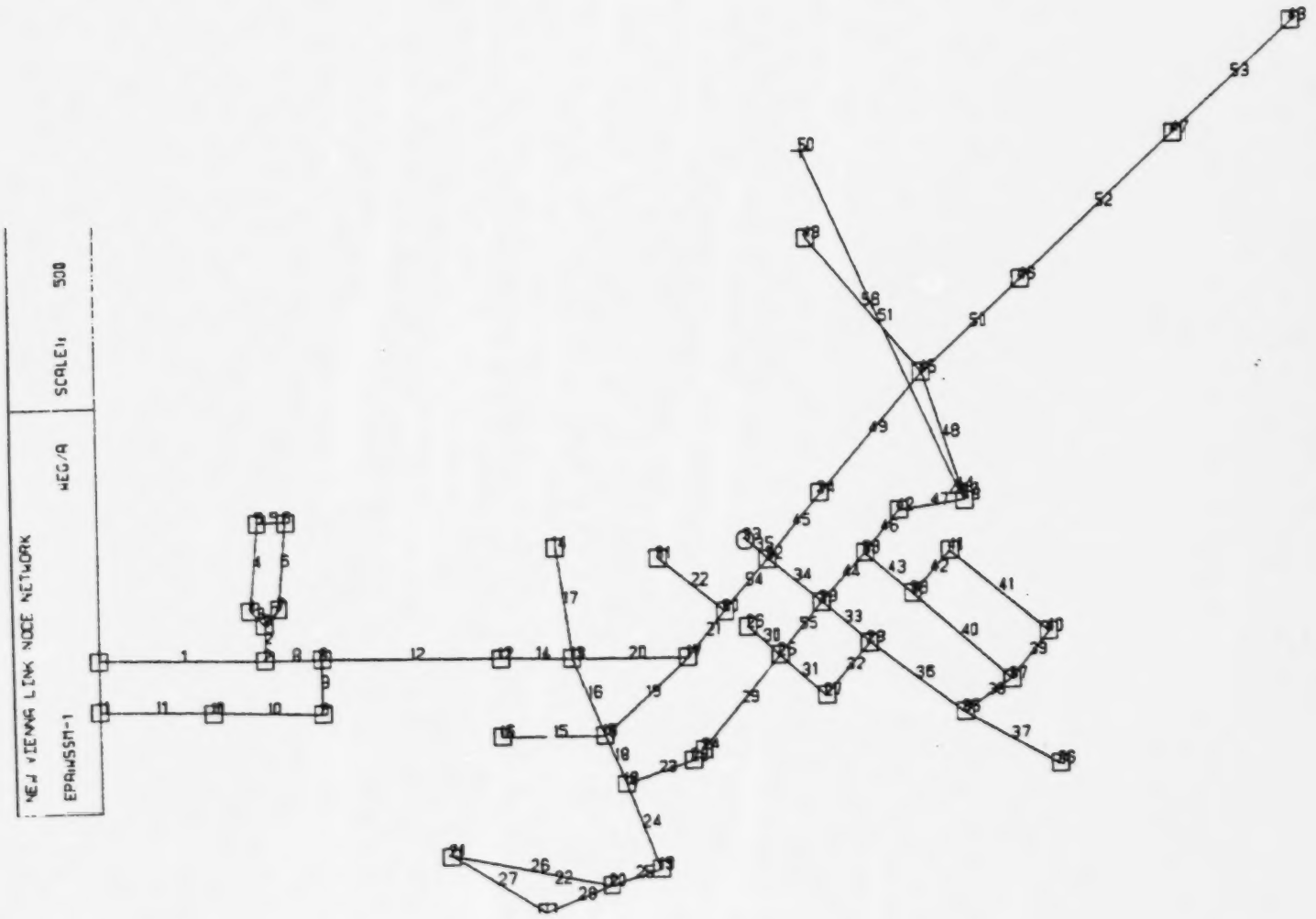



**Figure 21c: Sample Plot Output (original in color).**

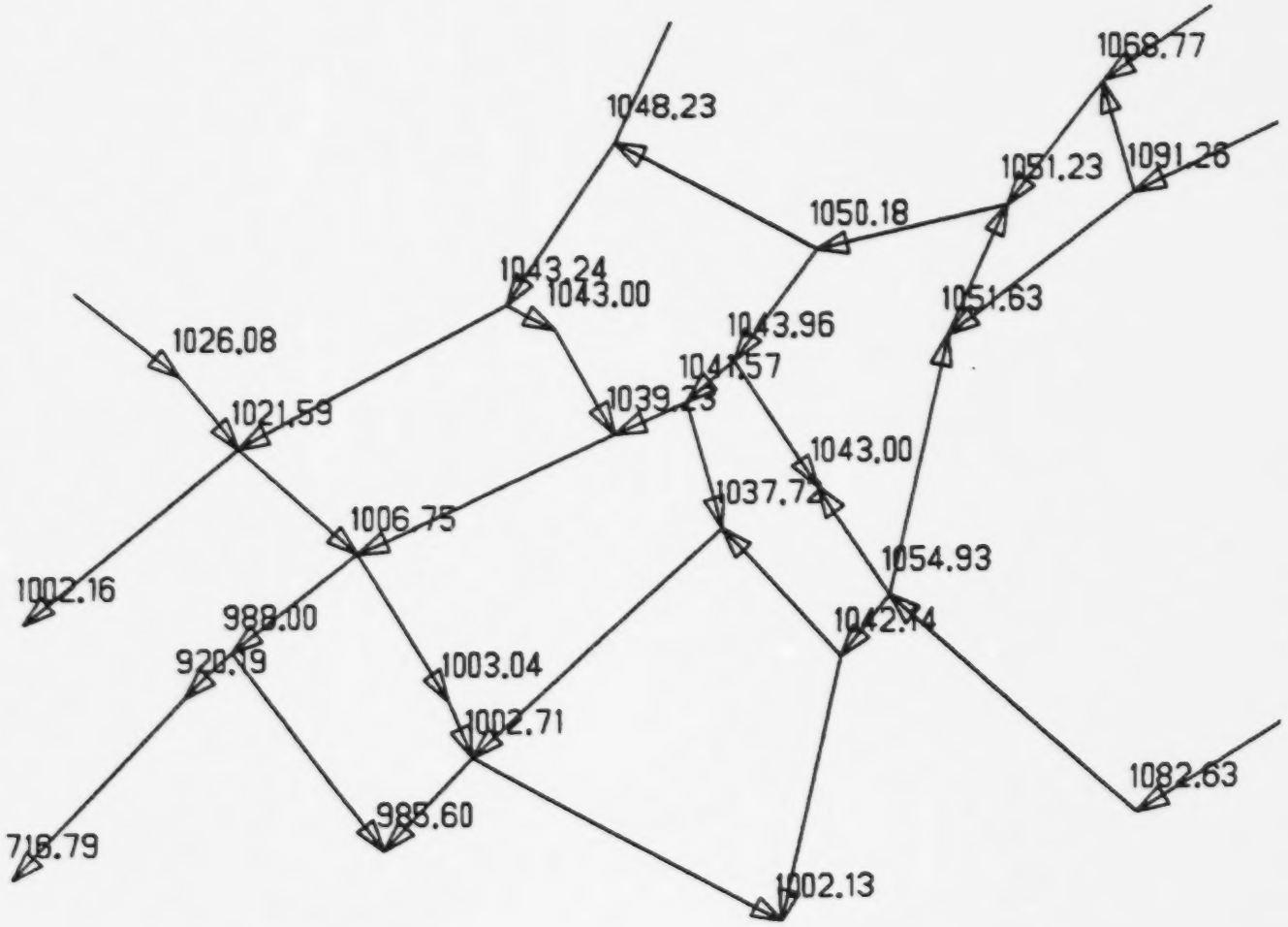



**Figure 21d: Sample Plot Output (portion).**

REPORTER PROGRAM

PURPOSE: Reporter generates tabular printouts of the selected data stored within the WSSM files.

OPTIONS: Both specific and general listing reports of the node and link files can be produced with the module. The user may: (1) select either file; (2) restrict listing to those records which satisfy a given description (i.e., pipes in servce zone 1, all pumps, all tanks, all demands within a specified range, etc.) or produce a general file listing; (3) select column or row oriented tables in final output.

RESTRICTIONS: With row oriented tables, a maximum of 10 items may be displayed and with column oriented data a maximum of 40 items may be displayed. Description of stored data is limited to 12 alpha characters. All data presented in tables is listed to two decimal place accuracy only.

OVERVIEW: It is frequently desirable to extract data from the node or link files in report format. The control switches provide the user with options for designing the type of report to generated. Careful planning of the control and selection cards will yield meaningful and useful reports. The input deck is designed to be simple to use.

INPUT DATA: The input data set can be separated into two card types: the control card and the storage location selection (SLS) cards.

CONTROL CARD

There are seven fields on the control card; 3 switch fields, 3 section fields, and the title field, described as follows:

| FIELD | CARD COLUMN | FORMAT | DESCRIPTION |
|---|---|---|---|
| 1 | 1 | I1 | File switch (1 = PIPE, 2 = NODE) |
| 3 | 4/5 | I2 | Selection switch (Ø = all records, 1 = equality test used for selection, 2 = range test used for selection) |
| 2 | 2/3 | I2 | Table format switch (Ø = column, $\geq$ 1 = row) |
| 4 | 6/10 | I5 | Field location within records for selection restriction test |
| 5 | 11/20 | F10.0 | Minimum acceptable value for selection by range test or the acceptable value for selection by equality test |
| 6 | 21/30 | F10.0 | Maximum acceptable value for range selection test |
| 7 | 31/70 | 10A4 | Output message. This alpha descriptor will appear within the page heading of each output page generated. |

| FIELD | CARD COLUMNS | FORMAT | DESCRIPTION |
|---|---|---|---|
| 1 | 1/2 | I2 | Field location within file of data to be reported. |
| 2 | 3/14 | 3A4 | 12 character alpha descriptor of contents of field |

SLS cards are required for each field desired. If more SLS Cards are provided than can be handled by the table format selected (10 for row, 40 for column), the program will terminate with a stop code.

PRINTER OUTPUT

REPORTER generates formatted report output on logical unit 6, and provides an echo of the input cards, and error messages, if any, on logical unit 10.

Below is a list of two error messages which may accompany an abnormal program termination of REPORTER.

"INPUT ERROR MIN ACCEPTABLE VALUE IS LARGER THAN THE MAXIMUM ACCEPTABLE VALUE"

"TOO MAY SPECIFIED COLS FOR LISTING OF RECORD"

ABNORMAL PROGRAM TERMINATION

ØØ32 If REPORTER terminates with a stop code of ØØ32, either message above will be printed on logical unit 10 indicating the source of the error.
0028 If REPORTER terminates with a stop code of ØØ28, the last selection card printed on logical unit 10 has an invalid storage location number.
0000 successful program termination.

Sample input and corresponding output for REPORTER are shown in Figures 22 and 23.

```
$RUN W.REPORTER.U 6=-OUTPUT   10=-TIME 11=NEWV2.DA.NODE 12=NEWV2.DA.PIPE
2 1 0                          SCENARIO COMPARISON REPORT
30 S1 COST
29 S3 COST
34 S1 PSI
47 S3 PSI
57 S1 CONC.
58 S3 CONC.
$ENDFILE
```

**Figure 22: REPORTER Sample Input (with Runner).**

```
 CONTROL CARD ECHO  2  1  0      0          0.0          0.0      SCENARIO COMPARISON REPORT
 IBYPAS=0, SELECTIVE LISTING DEACTIVATED
 SPECIFICATION CARD ECHO30    S1 COST
 SPECIFICATION CARD ECHO29    S3 COST
 SPECIFICATION CARD ECHO34    S1 PSI
 SPECIFICATION CARD ECHO47    S3 PSI
 SPECIFICATION CARD ECHO57    S1 CONC.
 SPECIFICATION CARD ECHO58    S3 CONC.
04-14-8216:26:40
```

Reproduced from best available copy.

**Figure 23a: REPORTER Sample Output (logical unit 10).**

```
PAGE      1                                          W S S M - 1
DATE 820414                                     NEW VIENNA WATER
TIME 162640                                SCENARIO COMPARISON REPORT
NODE FILE
 RECORDS                                                DESCRIPTIONS
              S1 COST      S3 COST      S1 PSI       S3 PSI       S1 CONC.     S3 CONC.
 REC NO   1    224.34       279.09       42.95        42.90          0.0          0.0
 REC NO   2     82.26       137.01       48.14        48.10          0.0          0.0
 REC NO   3     85.62       140.37       49.43        49.39          0.0          0.0
 REC NO   4    101.49       156.24       48.57        48.53          0.0          0.0
 REC NO   5    141.40       196.15       49.43        49.39          0.0          0.0
 REC NO   6    150.72       205.47       48.13        48.09          0.0          0.0
 REC NO   7     93.52       148.27       51.16        51.12          0.0          0.0
 REC NO   8     70.77       125.52       49.02        48.98          0.0          0.0
 REC NO   9     80.93       135.68       49.87        49.83          0.0          0.0
 REC NO  10    110.04       164.79       45.09        45.05          0.0          0.0
 REC NO  11    153.52       208.27       43.36        43.32          0.0          0.0
 REC NO  12     61.04       115.79       54.50        54.46          0.0          0.0
 REC NO  13     56.87       111.61       52.27        52.23          0.0          0.0
 REC NO  14     83.86       138.61       55.72        55.68          0.0          0.0
 REC NO  15     55.07       109.81       51.99        51.95          0.0          0.0
```

**Figure 23b: REPORTER Sample Output (logical unit 6). (Portion)**

SCANCOL PROGRAM

PURPOSE: SCANCOL generates a list of data in a particular storage column that does not fall within a user specified range.

OPTIONS: The user may specify the file to be scanned as well as an attribute restriction for selected searches.

RESTRICTIONS: None

OVERVIEW AND BACKGROUND: All working storage columns are set to '-9999' when the files are established via ESTBLINK: By specifying a value range which excludes this number or all negative numbers, a user can "scan" the particular column for records which failed to have data stored in this column. In general the program executes either of the following statements.

1. Print message if the record has value A1 in storage column A and does not have a value in column B within range

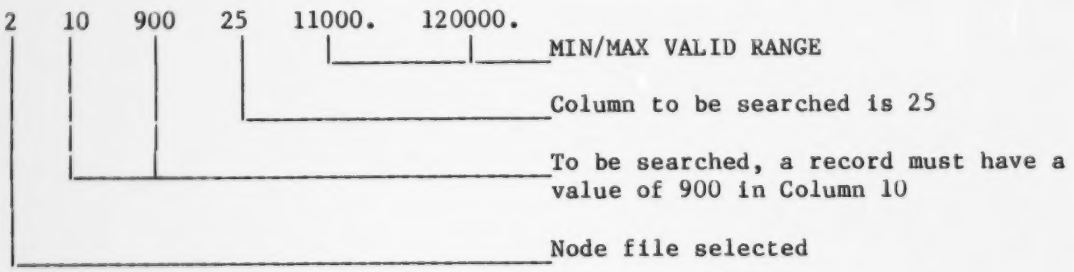
2. Print message if the record does not have a value in column B within range. Thus, the program can be used both to check on data entry, and to provide exception reports.

INPUT: The single input control card is read on logical unit 5 and has six fields as described below.

| FIELD | COLUMNS | FORMAT | DESCRIPTION |
|---|---|---|---|
| 1 | 1 | I1 | File Switch (1=Pipe 2=Node) |
| 2 | 2/4 | I3 | Selection switch ($\leq$0 No Restriction) $>$0:, value is column on which to base selection |
| 3 | 5/10 | I6 | Selection Value |
| 4 | 11/15 | I5 | Search Column |
| 5 | 16/25 | F10.0 | Minimum Value |
| 6 | 20/35 | F10.0 | Maximum Value |

No test is done by software to insure that the value in field 5 is less than that of field 6. Fields 1, 4, 5, and 6 are required for a scan. If fields 2 and 3 are left blank, the entire file will be scanned.

EXAMPLE INPUT:


2 10 900 25 11000. 120000.
MIN/MAX VALID RANGE
Column to be searched is 25
To be searched, a record must have a value of 900 in Column 10
Node file selected

STOP CODES

The following "stop codes" are generated by SCANCOL on termination:

ØØØØ - Successful search
ØØ24 - File switch improperly used
0028 - Restricted column out of range
ØØ32 - Search column out of range

A sample input and output of SCANCOL are shown in Figures 24 and 25.

```
$RUN W.SCANCOL.U 6=-OUTPUT 11=NEWV2.DA.NODE 12=NEWV2.DA.PIPE 10=-TIME
2              34   42.       50.
```

**Figure 24: SCANCOL Sample Input.**

```
04-14-8216:16:05
INPUT CARD ECHO 2   0     0    34       42.       50.
```

**Figure 25a: SCANCOL Sample Output (logical unit 10).**

```
SCANNING FOR      NEW VIENNA WATER        FILES ON 820414 AT 161605
NODE FILE SELECTED FOR SCAN
ENTIRE FILE TO BE SCANNED
COLUMN SCANNED IS  34 ACCEPTABLE RANGE BETWEEN       42.       50.
REC # VALUE  REC # VALUE  REC # VALUE  REC # VALUE  REC # VALUE  REC # VALUE
   7:    51.   12:    55.   13:    52.   14:    56.   15:    52.   16:    54.
  23:    51.   24:    52.   31:    52.   34:    54.   41:    51.   42:    51.
 TOTAL OUT OF RANGE IS    19
SCAN COMPLETED
```

**Figure 25b: SCANCOL Sample Output (logical unit 6). (Portion)**

SOLVER PROGRAM

PURPOSE: SOLVER is a general-purpose utility program capable of formulating and solving a variety of problems of general interest for water distribution systems, in particular for travel time to a node, concentration at a node, and cost of water at a node. The solution, consisting of values for each node, can be re-inserted into the data base under user control.

OPTIONS: Three types of solution can be specified:

a) concentration type - solving for the concentration of a constituent present in source nodes at defined quantities, at all other nodes in the network.
b) travel time type - solving for the total travel time to each node in the network, given initial conditions at source nodes.
c) cost type - solving for the total cost of delivered water at each node, based on flow-weighted averaging of costs, and assuming prior information as to cost factors associated with each individual node and link. Costs are presented as unit costs of delivered water at each node.

Other problems that can be formulated in the same framework as the cost, travel time, or concentration type solution, can also be solved.

RESTRICTIONS: Data cannot be inserted into restricted columns in the node file. A consistent hydraulic solution must be present in the data base prior to using SOLVER.

OVERVIEW: SOLVER is one of the key programs of the WSSM. It provides a simple method of solving a variety of problems of interest in analysis of water distribution systems. SOLVER assumes that a steady-state hydraulic solution has been obtained for the network, and that flows asociated with that solution have been stored in the data base. SOLVER then formulates a set of simultaneous linear equations for the desired solution type, based on continuity at each node, and assuming complete and uniform mixing at a node. SOLVER makes use of sparse matrix solution techniques, to solve the simultaneous equations. The code utilized was developed by Prof. M. Chandrashekar of the University of Waterloo, Ontario, and is used with his permission. The theoretical formualtion and development of equations underlying SOLVER has been presented in Volume I of this report.

Each of the three basic solution types requires that certain additional data be made available to the SOLVER program, as follows:

Type 1 - Concentration: The concentration of the parameter of interest at each node at which water enters the system must be specified. If the steady-state scenario shows that a tank is discharging to the network, then the parameter value must be specified for the tank as well. If it is desired to determine the percentage of water from any given source reaching a particular node, then that source can be given a parameter value of 100, and all other sources set to 0. SOLVER's solution will then directly provide the percentage of water from that source at all other nodes.

Type 2 - Travel Time: Again boundary conditions of the 'age' of water at all nodes at which water enters the system must be specified. In addition, the travel time for each link in the system must be previously calculated using program TRAVEL, and stored in the data base. The identical hydraulic scenario should be utilized in the TRAVEL and SOLVER programs. In considering tanks that discharge water into the system, the 'age' of the water in the tanks should either be estimated directly, or calculated based on hydraulic scenarios in which the tanks are filling.

Type 3 - Cost: In the cost solution, information on annualized costs for each link and node must be present and stored in the data base. As presently constituted, SOLVER cannot consider fixed and variable cost components for each element, but rather accepts only a single cost component, typically total annual cost. Again, boundary conditions must be specified for those sources discharging into the system. If tanks are discharging, then some value must be associated with the water in the tank, similar to the specification of an 'age' in the travel time type of solution, and this value can either be estimated or calculated through a tank-fill hydraulic scenario. In general, then, SOLVER requires the specification of values of the parameter being solved for at all nodes that are sources of water to the system, whether original sources, or tanks discharging to the system; in addition, depending upon the type of solution, exogenously determined link and node-related data (travel time, or cost) must also be specified. It is the user's responsibility to insure that all available data are present, and are determined consistent with the particular hydraulic scenario being examined.

PROGRAM INPUT:

Input to SOLVER consists of three cards, read on logical unit 5, specifying general run conditions, and card-image files on units 3 and 4, specifying 'override' conditions for node and link values respectively. The three cards read on logical unit 5 are a "type" card, a "scenario" card, and a "parameter" card. The type card indicates the type of solution to be performed; the scenario card defines the locations in the WSSM files in which the scenario and associated hydraulic values are stored; and the parameter card defines the columns in the WSSM files to be utilized to provide the supplementary information for node and link elements (e.g. travel time, costs).

It should be noted that SOLVER assumes that any nodal element with a scenario type of 0 is not present in the simulation, a type of 1 indicates that it is present as a demand node, and a type of 2 indicates that it is present as a source. In specifying a hydraulic scenario for WATSIM, a type of 1 indicates a fixed flow element, and a type of 2 indicates a fixed head (tank) element, which may either be a source or demand element. Thus, it may be necessary, depending upon the results of the hydraulic analysis, to alter the type coding for tank elements that are filling - otherwise, SOLVER will assume that they are sources of flow into the system.

The specific formats and meaning of the general input cards are given as follows:

| Card | Column | Format | Description |
|---|---|---|---|
| 1 (type) | 1 | I1 | Solution Type Selection Switch = 1 concentration<br>= 2 travel time<br>= 3 cost |
| 1 | 2 | I1 | Override Indicator = 0 no override data<br>= 1 node override data<br>= 2 link override data<br>= 3 node and link override |
| 2 (scenario) | 1/4 | I4 | Column in node file containing nodal type scenario |
| 2 | 5/8 | I4 | Column in node file containing nodal inflows and outflows (as developed by WATSIM) corresponding to the scenario selected. |
| 2 | 9/12 | I4 | Column in link file containing scenario link type |
| 2 | 13/16 | I4 | Column in link file containing link flows |
| 2 | 21/60 | 10A4 | Solver run title to appear on output |
| 3 (parameter) | | | |
| 3 | 1/4 | I4 | Nodal column containing boundary condition data |
| 3 | 5/8 | I4 | Link column containing parameter (cost or travel time) data |
| 3 | 9/12 | I4 | Column in node file in which output is to be stored (restricted to the range of 30 to 60). |
| 3 | 21/36 | 4A | Name of solution parameter (to appear on printed output) |

Note: for the travel time and concentration solutions, although a nodal column containing parameter data must be present, it is generally easier to set the column to a value containing all 0., and use the override node deck to set the boundary conditions for source nodes. For the cost solution, the nodal parameter column should contain the annualized nodal costs. The 'override' decks for the node and link files have an identical format. One card per element override is used, as follows:

| Card Column | Format | Description |
|---|---|---|
| 1/5 | I5 | node or link number |
| 11/20 | F10.0 | parameter value |

As noted above, the node override deck is presented on logical unit 3, and the link override deck is presented on logical unit 4.

PROGRAM OUTPUT:

Program output consists of an output data set on logical unit 6, containing an echo of run conditions, and an output on logical unit 10, containing a printed table of solution values at each node. In addition, data may be inserted into the WSSM node file storage column, at the election of the user, as described above.

Sample input and output of SOLVER, run in the cost mode, is presented in Figures 26 and 27.

```
$RUN W.SOLVER.O 10=-TIME 6=-OUTPUT 3=W.SOLVS1.DAT 11=NEWV2.DA.NODE 12=NEWV2.DA.PIPE
$1
  31  35  36  38      ANNUAL $/YR PER GPM COST $1
  41  26  60$/YR PER GPM
$ENDFILE
```

**Figure 26a: SOLVER Sample Input.**

```
  36       50.
  55      8969.95
```

**Figure 26b: NODE Override Deck Input (logical unit 3).**

```
 TYPE  3 SOLUTION METHOD ON 820219 AT  94322 FOR      NEW VIENNA WATER
NODE   CONTROL   COLS:     31    35 LINK   CONTROL   COLS:     36    38 FOR    ANNUAL $/YR PER GPM COST $1
NODE INPUT COL=  41 LINE INPUT COL:  26 STORAGE COLUMN 60 VARIABLE DESCR: $/YR PER GPM
NUMBER OF PIPES MODELED =    55
 NUMBER OF NODES I O  BE ORDERED    49

                                          NEW VIENNA WATER
DATE: 820219                            ANNUAL $/YR PER GPM COST $1
TIME:  94322        NODE SPECIFICATIONS     RESULTS OF ANALYSIS              PAGE  1
                                        NODE  $/YR PER GPM
                                         NO
                                          1          294.78
                                          2          108.09
                                          3          112.51
                                          4          133.36
                                          5          185.80
                                          6          198.05
                                          7          122.89
```

**Figure 27a: SOLVER Sample Output (logical unit 6). (Portion)**

```
BEGINING NODE VALUES OVERRIDE INSERTION
 NODE=    36 VALUE=      50.00
 NODE=    55 VALUE=    8969.00
```

**Figure 27b: SOLVER Output (logical unit 10).**

TRAVEL PROGRAM

PURPOSE: TRAVEL calculates the travel time from one end of a link to the other, based on stored distance and velocity data in the link file.

OPTIONS: None

RESTRICTIONS: It is the user's responsibility to see that velocity and length measurements are consistent. If a stored velocity is zero in value then the travel time for that link is set to zero.

OVERVIEW: TRAVEL is used as an intermediate step in the process of determining the travel time from a source to a specific node. It is generally run after a specific hydraulic scenario has been inserted into the data files, and must be run prior to using the SOLVER program in travel time mode. TRAVEL operates simply on a link by link basis, dividing the stored distance by the stored velocity, to calculate the travel time. As velocity is a signed quantity, travel time is taken as distance divided by the absolute value of velocity.

CARD INPUT:

Card input for TRAVEL is read on logical unit 5. A single card, containing the columns in the link file in which velocity is stored and travel time is to be stored, is required:

| Field | Format | Card Column | Decription |
|---|---|---|---|
| 1 | I2 | 1/2 | Column in link file for insertion of travel time data (must be larger than 20). |
| 2 | I2 | 4/5 | Column in link file containing velocity data |

OUTPUT:

TRAVEL inserts data into the link file, and provides a printed record of the calculations performed for each link. If a link has a specified velocity of zero, a message is printed on the output listing.

Sample input and output for program TRAVEL is shown in Figures 28 and 29.

```
$RUN W.TRAVEL.U 6=-OUTPUT 10=-TIME 11=NEWV2.DA.NODE 12=NEWV2.DA.PIPE
33 39
$ENDFILE
```

**Figure 28: TRAVEL Sample Input (with Runner).**

```
 TIME OF TRAVEL CALCULATIONS PERFORMED
              ON  820414 AT   144810
         FOR  NEW VIENNA WATER SYSTEM
DATA TO BE STORED IN COLUMN 33
 VELOCITIES READ FROM COL:   39
LINK          TIME      VELOCITY DISTANCE
   1      52000.000      0.020   1040.000
   2        777.385      0.283    220.000
   3        863.309      0.139    120.000
   4       5612.242      0.098    550.000
   5      10000.000      0.018    180.000
   6       5339.805     -0.103    550.000
   7        967.742     -0.124    120.000
   8        849.056      0.424    360.000
   9        778.302      0.424    330.000
  10       2310.231      0.303    700.000
  11       5950.410      0.121    720.000
  12       1141.414      0.990   1130.000
  14        406.720      1.131    460.000
  15      10819.672      0.061    660.000
  16        834.646     -0.635    530.000
  17       4320.984      0.162    700.000
  18        466.761      0.707    330.000
  19        771.704      0.933    720.000
  20        740.000      1.000    740.000
  21        404.157      0.866    350.000
  22       2722.772      0.202    550.000
  23       3719.008     -0.121    450.000
  24       1415.094      0.424    600.000
  25       2828.283      0.099    280.000
  26      45000.000      0.020    900.000
 TIME OF TRAVEL FOR NEW VIENNA WATER SYSTEM COMPLETE
```

Reproduced from best available copy.

**Figure 29: TRAVEL Sample Output (partial).**

WATINT PROGRAM

PURPOSE: WATINT prepares the node and link card-image decks necessary to run the hydraulic analysis model WATSIM, by extracting relevent data from the WSSM node and link files.

OPTIONS: None

RESTRICTIONS: WATINT prepares only the node and link card-image decks necessary for WATSIM - for each run of WATSIM, the header control cards determining the WATSIM options to be used must be added separately. WATINT is restricted to steady-state mode runs of the WATSIM model.

OVERVIEW: WATINT takes as input a description of the record positions (columns) in the node and link files containing the scenario description and associated values, and generates a card-image data file in a format consistent with WATSIM input. The scenario columns specify which nodes and links are to be included in the particular model run, and how they should be represented in the model (e.g. for nodes, as a fixed head or fixed flow node, and for pipes, as a pipe, pump element, etc.) The user must design the scenario appropriately to insure that it is a proper representation of the situation to be modeled. Other WSSM programs (NCOPYATT, LCOPYATT, INSEND, AND INSELK) aid the user in placing a scenario into the data base.

SCENARIO REPRESENTATION:

1) Nodes

The following node types are permitted in describing a scenario:

| Node Type | Representation |
|---|---|
| 0 | Node is excluded from a scenario |
| 1 | Node is included, treated as fixed flow (source or demand) |
| 2 | Node is included, treated as fixed head (tank or reservoir) |

All nodes specified as type 1 are expected to have flow values available in the value column. For nodes specified as type 2, the information in the value column is expected to be the head in feet.

Two columns are required to specify the nodal information for a scenario - the first column defines the nodal types, and the second the values to be associated with the types.

2) Links

While only a single column is used to specify the link scenario, other information is expected to be in standard locations in the link file, as shown in the table of link file record information. The following link types are valid:

| Type | Representation |
|---|---|
| 0 | Link is excluded from scenario |
| 1 | Link is represented as ordinary pipe |
| 2 | Link is represented as closed pipe |
| 3 | Link is represented as check valve |
| 4 | Link is represented as pressure regulating valve |
| 5 | Link is represented as in-line booster pump |
| 6 | Link is represented as parabolic pump |

It should be noted that it is the user's responsibility to see that all required information abut a scenario is both present and consistent for the link and node files. For in-line booster pump elements, the lift in feet must be present in column 29. For pressure regulating values, the head increase in feet must be present in column 29 for that record for a parabolic pump. For a parabolic pump, coefficients A, B, and C, as expected by WATSIM must be present in columns 26, 27, and 28, respectively.

CARD INPUT:

Input to WATINT is expected on logical unit 5. Two control cards, the link and node cards, are required.

Card 1: Node Control Card

| Field | Column | Format | Description and Comments |
|---|---|---|---|
| 1 | 1/5 | I5 | Column containing node types for scenario |
| 2 | 6/10 | I5 | Column in node file containing node values for scenario |
| 3 | 11/20 | F10.0 | The fixed increment, in feet, to be added to the ground elevation of all type 1 nodes to estimate the head of that node for WATSIM. The default, if the field is left blank, is 100'. |

Card 2: Link Control Card

| Field | Column | Format | Description and Comments |
|---|---|---|---|
| 1 | 1/5 | I5 | Column in link file containing link type scenario. |

An example of sample input to WATINT is shown in Figure 30.

PRINTER OUTPUT:

Messages are displayed on logical unit 6. These messages show the run date, and echo the input cards. An example is shown in Figure 31.

CARD IMAGE OUTPUT:

The node and line input decks for WATSIM are generated as a card-image data file on logical unit 9. Three WATSIM control cards must be prefixed to this card-image file to obtain a valid WATSIM input data file. An example of a complete WATSIM input data file, showing the control cards and the portion generated by WATINT, is shown in Figure 32.

```
$RUN W.WATINT.U 6=-OUTPUT3 10=-TIMES 11=NEWV2.DA.NODE 12=NEWV2.DA.PIPE 9=W.NVS2
   31   33
   36
$ENDFILE
```

**Figure 30: WATINT Sample Input.**

```
ENTER NODE COLUMNS - SCENARIO/FLOWS-HEADS/ELEV.INC  (DEFAULT=100FT.) IN 215,F10.0
WATSIM LINE AND NODE DECK
ON 820203 AT 164/02 FOR       NEW VIENNA WATER          FILE
USING NODE COLUMNS    31   33 ELEVATION INC.=    100.0000
ENTER COLUMN FROM LINK FILE FOR SCENARIO (I5)
AND LINK COLUMN    36
END DECK PROCESSING
```

**Figure 31: WATINT Printer Output (logical unit 6).**

```
NQ      32      1    0.0     1128.    1228.      0.      0.
NH      33      1    0.0     1125.    1235.    1234.   1236.
NQ      34      1    0.0     1116.    1216.      0.      0.
NQ      35      1    0.0     1135.    1235.      0.      0.
NQ      36      1    0.0     1139.    1239.      0.      0.
NQ      37      1    0.0     1138.    1238.      0.      0.
NQ      38      1    0.0     1134.    1234.      0.      0.
NQ      39      1    0.0     1135.    1235.      0.      0.
NQ      40      1    0.0     1132.    1232.      0.      0.
NQ      41      1    0.0     1122.    1222.      0.      0.
NQ      42      1    0.0     1121.    1221.      0.      0.
NQ      43      1    0.0     1125.    1225.      0.      0.
NH      44      1    0.0     1125.    1245.    1244.   1246.
NQ      45      1    0.0     1136.    1236.      0.      0.
NQ      46      1    0.0     1132.    1232.      0.      0.
NQ      47      1    0.0     1160.    1260.      0.      0.
NQ      48      1    0.0     1177.    1277.      0.      0.
NQ      49      1    0.0     1125.    1225.      0.      0.
88
L       2       1     1      1        1040.    4.00   130.00    0.0          1
L       2       1     3      1         220.    4.00   130.00    0.0          2
L       3       1     4      1         120.    4.00   130.00    0.0          3
L       4       1     5      1         550.    4.00   130.00    0.0          4
L       5       1     6      1         180.    4.00   130.00    0.0          5
L       6       1     7      1         550.    4.00   130.00    0.0          6
L       7       1     3      1         120.    4.00   130.00    0.0          7
L       8       1     2      1         360.    4.00   130.00    0.0          8
L       8       1     9      1         330.    4.00   130.00    0.0          9
L       9       1    10      1         700.    4.00   130.00    0.0         10
L      10       1    11      1         720.    4.00   130.00    0.0         11
```

**Figure 32: WATINT Card-Image Output (logical unit 9). (Portion)**

WATWSSM PROGRAM

PURPOSE: WATWSSM is an interface program, used to insert flow, velocity, pressure, and head data developed through a run of WATSIM into the appropriate node and link records of the WSSM.

OPTIONS: Any or all of the data elements generated by WATSIM for the node and links may selectively be placed in the data base.

RESTRICTIONS: Data may not be inserted into protected columns of the node and link files. Protected columns for the node file are columns 1, and 5 through 17; for the link file, protected columns are 1, and 5 through 12.

OVERVIEW:

WATSIM, as modified for use in the WSSM, generates a card image data file containing, for each node, the WSSM node number, and the flow, pressure, and head at that node, as determined by the hydraulic solution. Similarly, additional card images are generated for each link, containing flow and velocity data. Thus, there are potentially three data elements to be inserted in the node file, and two in the link file, for each record. WATWSSM allows the user to control and direct this data insertion to user-specified columns in the node and link file.

PROGRAM INPUT:

A control card is read on logical unit 5, and the card-image summary output from WATSIM is read on logical unit 14. As there is no modification to the WATSIM card-image file, only the control card, specifying the insertion process to the node and link files, is described.

CONTROL CARD:

| Field | Format | Column | Description |
|---|---|---|---|
| 1 | I3 | 1/3 | Column in node file for pressure insertion |
| 2 | I3 | 4/6 | Column in node file for flow insertion |
| 3 | I3 | 7/9 | Column in node file for head insertion |
| 4 | I3 | 10/12 | Column in link file for flow insertion |
| 5 | I3 | 13/15 | Column in link file for velocity insertion |

If any of the specified columns is less than zero, then no insertion will take place for that data element for any records. If any of the nodal columns is in a protected field, then no nodal insertion for any data will take place, and similarly for the link columns.

PROGRAM OUTPUT:

Printed output consists of messages echoing the run conditions, and detecting errors.

An example of program input and output for program WATWSSM is shown in Figures 33 and 34.

```
$RUN W.WATWSSM.0 6=-OUTPUT 10=-TIME 11=NEWV2.DA.NODE 12=NEWV2.DA.PIPE 14=-DEK10UT
 34 35 36 38 39
$ENDFILE
```

**Figure 33: WATWSSM Sample Input (with Runner).**

```
DATA BASE INSERTION PROCESSING FOR HYDRAULIC MODEL ON   820203 A1   172714
NODE PRESSURE INSERTION COLUMN:    34
 NODE FLOW INSERTION COLUMN:    35
 NODAL HEAD INSERTION COLUMN   36
 LINK Q INSERTION COLUMN:    38
 LINK VELOCITY INSERTION COLUMN   39

   55 FLOWS INSERTED INTO FILE
END INSERTION PROCESSING FOR HYDRAULICS
```

**Figure 34: WATWSSM Printer Output.**

TABLE 1

ASSIGNMENT OF LOGICAL UNITS TO PROGRAM MODULES

The standard usage of logical units for all prograzms of WSSM is as follows:

| Logical Unit | Usage |
|---|---|
| 5 | Card Image Input |
| 6 | Primary Printed Output |
| 10 | Secondary Printed Output |
| 11 | Node File |
| 12 | Link File |

Deviations from this scheme are as follows:

| Program | Supplemental Information on Logical Units |
|---|---|
| ERCHECK | Does not use logical unit 5 |
| NAMEIN | Does not use logical unit 10 |
| NETPLOT | Does not use logical unit 10<br>Also provides plot data output file on unit 9 |
| WATINT | Places card image output file on unit 9 |
| WATWSSM | Expects card-image input data<br>(output of WATSIM) on unit 14. |

In addition, the logical unit requirements for program WATSIM (not described in this documentation) are as follows:

| Logical Unit | Usage in WATSIM |
|---|---|
| 5 | Card-image input |
| 6 | Printer output |
| 12 | Card-image output |
| 15 | Work file |
| 17 | Work file |

TABLE 2

UPDATE CODES

Stored in Word 11 of link and node header records

| | |
|---|---|
| ESTBLINK | 1040 |
| ESTBNODE | 1020 |
| FIXL | 2040 |
| FIXN | 2020 |
| INSELK | 3040 |
| INSEND | 3020 |

## PROGRAMMER'S GUIDE

The WSSM system consists of a set of data files and programs which create, edit, modify, display, and act upon these files. As currently constituted, WSSM is designed to run on any large-scale computer with minor modifications between sites. WSSM is programmed in FORTRAN, and makes extensive use of direct access programming techniques. It is assumed, in the remainder of this guide, that the WSSM programmer is familiar with FORTRAN direct access programming, and is aware of the basic WSSM concepts explained in Volume 1. This guide is not a tutorial, but rather a reference source to supplement training, and information contained in the source code.

### FILE ORGANIZATION

A WSSM data base describes the water supply network it models. There are two files which constitute the WSSM data base, the LINK file and the NODE file. All operations of WSSM are carried out on these two files. The LINK and NODE files are interrelated, and taken together constitute a description of the topology, location, and data attributes of the water supply system being modeled. Certain information is stored redundantly in both files, for ease in programming.

The LINK and NODE files are organized similiary, differing only in record size and the type of data being stored. Each has a header record followed by a number of physical records. Each physical record contains 10 logical records. The exact number of physical records (including the header record) is stored within the header record as is the largest number of logical records that can be stored within the file. The physical record length of the LINK file is 400 words while that of the node file is 600 words. Thus the header record of the

LINK file is 400 words and that of the NODE file is 600 words. The length of the logical record of the LINK file is 40 words, and is 60 words for the NODE file. Each logical record in the LINK file corresponds to a pipe in the water system being modeled. Each logical record in the NODE file corresponds to a specific component (junction, tank, demand) of the modeled system. WSSM currently has the capacity to model up to 790 pipes and 790 nodes.

DATA STRUCTURE

The information stored in the LINK and NODE header records is shown in Tables 3 and 4. The information stored in the LINK and NODE logical records is shown in Tables 5 and 6. As can be seen from Table 5, specific columns within the link record are reserved (may only be modified by modules FIXL and FIXN). Word positions 1 to 25 are also partially protected in that the modeling programs have been coded to prohibit data entry into these locations. Similary, in the node logical record, specified columns are reserved and word positions 1 to 29 are partially protected from data entry by the modeling modules.

LOGICAL RECORD ACCESS METHODOLOGY

From the programmer's point of view, almost all interface with the data files is done through intermediate access subroutines, which handle the conversion between logical and physical records. Thus, the programmer deals in terms of the header record and the logical records. The access subroutines perform all necessary conversions. It is most important, however, that the programmer understand the nature of the access subroutines or else errors will occur. The basic file access routines are OPEN, HEAD, GETLK, GETND, PUTLK, PUTND, and CLOSE. Subprograms OPEN and HEAD are necessary for all file access, and CLOSE is necessary to all programs which enter data into the data base. GETLK and

TABLE 3.

LINK HEADER RECORD

| Word | Description |
|---|---|
| 1 | Highest legal record number |
| 2 | Date file established |
| 3 | Time file established |
| 4 | Date of last update by editor module program |
| 5 | Time of last update by editor module program |
| 10 | Highest currently used logical record number |
| 11 | code for last program that modifies file |
| 12 | number of physical records |

All other words - currently unused, but reserved for future use.

TABLE 4

NODE HEADER RECORD

| Word | Description |
|---|---|
| 1 | Highest legal logical record number |
| 2 | Date file established |
| 3 | Time file established |
| 4 | Date of last update by editor module program |
| 5 | Time of last update by editor module program |
| 6 | Bounding rectangle: Y maximum - north |
| 7 | Y minimum - south |
| 8 | X maximum - east |
| 9 | X minimum - west |
| 10 | Highest currently used logical record number |
| 11 | Code for last program that modified file |
| 12 | Number of physical records |
| 10/17 | File name |

All other fields - currently unused, but reserved for future use.

TABLE 5

LOGICAL LINK RECORD

| Word | Description |
|---|---|
| * 1 | ID |
| 2 | Type |
| 3 | Service Zone |
| 4 | Secondary Zone |
| * 5 | Node 1 (upstream) |
| * 6 | Node 2 (downstream) |
| * 7 | X Coordinate at Node 1 |
| * 8 | Y Coordinate at Node 1 |
| * 9 | Z Coordinate at Node 1 |
| * 10 | X Coordinate at Node 2 |
| * 11 | Y Coordinate at Node 2 |
| * 12 | Z Coordinate at Node 2 |
| 13 | Diameter |
| 14 | Length |
| 15 | Date of installation or replacement |
| 16/20 | Working storage |
| 21 | Frictional coefficient (C-factor) |
| 22/40 | Model specific attributes/working storage |

*Protected field

TABLE 6

LOGICAL NODE RECORD

| Word | Description |
|---|---|
| * 1 | ID |
| 2 | Type |
| 3 | Service zone |
| 4 | Secondary Zone |
| * 5 | X (Easting) |
| * 6 | Y (Northing) |
| * 7 | Z (elevation) |
| * 8/17 | ID's of links connected to nodes |
| 18/60 | Nodal specific attributes/working storage |

*Protected field

PUTLK are the subroutines that retrieve and store logical LINK records, and GETND and PUTND are the subroutines that retrieve and store logical NODE records. Inasmuch as the access methods are analogous for each (with the exception of the difference in logical and physical record lengths), only the link file will be discussed.

The access methodology makes use of a buffering system, consisting of a single in-core buffer, which significantly cuts down the number of direct access input/outputs, particularly in sequential file processing. A logical flag is set to indicate whether a program has made any change to this in-core buffer. If the core buffer has not been modified, then the flag is false, otherwise it is true. The status of the buffer update flag, and the physical record number of the buffer in core, are communicated between OPEN, GETLK, PUTLK, and CLOSE through a named common block.

When a file is to be accessed for the first time within a program, OPEN and HEAD are called. OPEN initializes the logical flag to false and sets the physical record number to 1. HEAD retrieves the file size parameters from the header record. When the program wants to obtain the information associated with a link, a call to GETLK for that link number is issued. GETLK first calculates the physical record number associated with the link record, and determines whether or not that physical record is currently located in the buffer. If the record is in the buffer, the subroutine accesses the appropriate logical record and returns the 40 word "link array". If the appropriate physical record is not in core, the subroutine first checks to see if the buffer has been updated previously (i.e. a change made in one or more of the link records stored in the buffer) by examining the logical flag. If the buffer has not been updated, GETLK simply over-writes it with the new physical record containing the link record of interest, obtained from the file. If, however, the

buffer has been updated, GETLK first writes the buffer back onto disk prior to obtaining the new physical record. When GETLK brings the new physical record into the buffer, it reinitializes the logical flag to false, and the new physical record number is retained in the named common block as the "current" physical record held in the buffer.

Subroutine PUTLK is the reverse of GETLK in that it is designed to enter updated logical link records into the link file. When a change is made to a logical link record it is made to the corresponding 40 word array within the program and passed to PUTLK via its argument list. If the appropriate physical record is in the buffer, PUTLK updates the buffer and sets the logical flag to true. If a different physical record is curently in the buffer, PUTLK checks the update flag. If it is true, then the physical block currently in the buffer is written to the link file and the new physical record which contains the link record of interest is retrieved from the link file. If it is false, then the current physical record is simply overwritten with the new, and the appropriate changes are made to the logical flag and the physical record number.

Subroutine CLOSE is used to insure that if the buffer has been updated, it is written back to disk. Thus, the only subroutine that can actually modify the buffer in core is PUTLK, but GETLK and CLOSE as well as PUTLK all can create a situation in which the buffer is written back to the file. If CLOSE is not called at the end of a program which has made modifications to the files, then the last buffer will not be written back to the file. Abnormal program termination will also cause the in-core buffer to be lost. In both cases, the file will not properly reflect the desired changes.

In order to modify a link record, the array must be modified first and then passed to PUTLK. All calls to PUTLK should be preceded by a call to GETLK with modifications to the link array occuring between the calls. It

should be noted that the I/O modules themselves do not prevent the user from destroying data stored in reserved record locations, thus care must be exercised in any modifications to the link array.

In general the link file is provided on logical unit 12 and the node file on logical unit 11. The necessary Define File statements and assignment of logical unit device numbers are contained in subroutines OPEN and HEAD respectively.

The data retrieval and data entry subroutines return a status code which indicates the level of success of the effort. A return code of zero is the normal return. If the return code is not zero, the contents of the record array are not predictable. Consequently all calls to GETLK, GETND, PUTLK, and PUTND should be followed by a check on the return code, and a branch to the appropriate error handling location. For sequential processing, it is sufficient to ignore all non-zero return codes and loop to access the next logical record.

Table 7 lists a series of subprograms that may be considered as utility routines of WSSM. Each subprogram is discussed separately. The named common blocks are described in a later section of this document.

TABLE 7

SUBPROGRAMS

| | |
|---|---|
| ADIST | HEADL |
| BCHECK | INHEAD |
| BOUND | INITL |
| BRWRIT | NAME |
| CLOSE | OPEN |
| COLCHK | PUTLK |
| COMPAR | PUTLKZ |
| CONCHK | PUTND |
| COVER | PUTNDZ |
| DATIME | UHEAD |
| GETLK | ULHEAD |
| GETND | UPDATN |
| HEAD | |

ADIST SUBPROGRAM

PURPOSE: To calculate the distance between two points.

CALLING SEQUENCE: D = ADIST (X1, Y1, X2, Y2)

COMMON BLOCKS: NONE

ARGUMENT LIST:

| | |
|---|---|
| X1 | Easting, first coordinate |
| Y1 | Northing, first coordinate |
| X2 | Easting, second coordinate |
| Y2 | Northing, second coordinate |
| D | Distance between point (XI, Y1) and (X2, Y2) |

BCHECK SUBPROGRAM

PURPOSE: Used in FIXN: tests new coordinates against current bounding rectangle: will expand bounding rectangle if the new point falls outside the bounding rectangle window.

CALLING SEQUENCE: Call BCHECK (COORD,LOGICL)

COMMON BLOCKS:

/ RECT / XMAX, XMIN, YMAX, YMIN
/ UBR / UBRL

ARGUMENT LIST:

COORD 3 word vector containing the new coordinates
1 easting
2 northing
3 elevation (not used in this module)

LOGICL 3 word logical vector
word position I must be set to .TRUE. if a non-zero value is contained in word position I of COORD.

BRWRIT SUBPROGRAM

PURPOSE: To print out the bounding rectangle in an easily recognizable form.

CALLING SEQUENCE: Call BRWRIT (ANHEAD,LUNIT)

COMMON BLOCKS: None

ARGUMENT LIST:

ANHEAD - The header record of the node file; ANHEAD is dimensioned to 600.

LUNIT - Logical unit assigned to the printer.

CLOSE SUBPROGRAM

PURPOSE: Writes final updated buffers to the node and link files.

CALLING SEQUENCE: Call CLOSE

COMMON BLOCKS:

/LHEAD/ - LDEV,JMAX,JBLOCK,ALHEAD(400)
/DAEH/ - NDEV,IMAX,IBLOCK,ANHEAD(600)
/UPLINK/ - PIPEBK,NL,UPDATL
/UPNODE/ - VERTBK,NN,UPDATN

ARGUMENT LIST: None

COLCHK SUBPROGRAM

PURPOSE: To check the columns for data insertion against a list of write protected columns in the LINK file.

CALLING SEQUENCE: Call COLCHK(ICOL,ISTAT,NTOT)

COMMON BLOCKS: None.

ARGUMENT LIST:

ICOL - vector containing columns in which insertion is to take place dimensioned (6)

ISTAT - counts number of write protected columns that are listed in vector ICOL

NTOT - is returned by COLCHK: NTOT is a counter of the number of columns in which insertion is to take place.

COMPAR SUBPROGRAM

PURPOSE: Used in ERCHECK, this module compares the geographic coordinates of the node record to the geographic data stored in the link record.

CALLING SEQUENCE: Call COMPAR(ILINK,INODE,GEO,ANODE,ICNT,ISTAT)

COMMON BLOCKS:

/LUNIT/ IREAD, IRITE

ARGUMENT LIST:

ILINK - Link record numeric ID
INODE - Node record numeric ID
ANODE - 60 Word vector containing the node logical record
GEO - 3 Word vector containing the coordinates of the link record
ICNT - Returned by COMPAR, number of links connected to node INODE
ISTAT - Returned by COMPAR, equal to the return code from the I/O module GETND

COVER SUBPROGRAM

PURPOSE: To make the necessary cross over corrections in the node file associated with addition, update, deletions to the link file.

CALLING SEQUENCE: Call COVER (ANODE,ILK,N,ISTAT)

COMMON BLOCKS:

/LUNIT/ - IREAD,IRITE

ARGUMENT LIST:

ANODE - 60 word vector containing the node logical record
ILK - link upon which corrections were made
N - update switch, negative for deletion from connectivity list
positive for insertion into connectivity list
ISTAT - completion status code
=0 successful crossover corrections
=1 connectivity switch passed was zero/indeterminate; requested corrections aborted
=2 pipe is not inserted into connectivity list for lack of room

DATIME SUBPROGRAM

PURPOSE: To return to the main module the current date and time in packed code format.

CALLING SEQUENCE: Call DATIME (NDATE,NTIME)

COMMON BLOCKS: None

ARGUMENT LIST:

NDATE - 6 digit packed decimal number of form YYMMDD (year, month, day)
NTIME - 6 digit packed decimal number of form HHMMSS (hour, minute, second)

NOTE: This subroutine must be interfaced with the date/time subroutines of the operating system being used. On the Michigan Terminal System, the date/time format is such that DATIME uses a temporary file on logical unit 10 to reformat the values into a packed code.

GETLK SUBPROGRAM

PURPOSE: Deliver a link record from the link file

CALLING SEQUENCE: Call GETLK(APIPE,ILK,ISTAT)

COMMON BLOCKS:

/UPLINK/- PIPEBK,NL,UPDATE
/LHEAD/ - LDEV,JMAX,JBLOCK,ALHEAD(400)

ARGUMENT LIST:

APIPE - 40 word link record obtained from file
ILK - link number of desired record
ISTAT - return status code =0 normal
=1 ILK out of file range
=2 ICK ≠ APIPE(1)
=3 I/O operation error

GETND SUBPROGRAM

PURPOSE: Deliver a node record from the node file

CALLING SEQUENCE: Call GETND(ANODE,IND,ISTAT)

COMMON BLOCKS:

/UPNODE/- VERTBK,NN,UPDATN
/DAEH/ - NDEV,JMAX,IBLOCK,ANHEAD(600)

ARGUMENT LIST:

ANODE - 60 word link record obtained from file
IND - node number of desired record
ISTAT - return status code =0 normal
=1 IND out of file range
=2 IND ≠ ANODE(1)
=3 I/O operation error

HEAD SUBPROGRAM

PURPOSE: To bring into core necessary information from the node and link header records.

CALLING SEQUENCE: Call HEAD(NLINK,NNODE)

COMMON BLOCKS:

/RECT/ - XMAX,XMIN,YMAX,YMIN
/LHEAD/ - LDEV,JMAX,JBLOCK,ALHEAD (400)
/DAEH/ - NDEV,IMAX,IBLOCK ANHEAD (600)

ARGUMENT LIST:

NLINK - Largest currently used link ID
NNODE - Largest currently used pipe ID

HEADL SUBPROGRAM

PURPOSE: Used in ESTBLINK, updates the link and node header records

CALLING SEQUENCE: Call HEADL (NDATE,NTIME)

COMMON BLOCKS:

/LHEAD/ - LDEV,JMAX,JBLOCK,ALHEAD(400)

ARGUMENT LIST:

NDATE - current date
NTIME - current time

INHEAD SUBPROGRAM

PURPOSE: Used in ESTBNODE, updates the link and node header records

CALLING SEQUENCE: Call INHEAD(NDATE,NTIME)

COMMON BLOCKS:

/RECT/ - XMAX, XMIN, YMAX, YMIN
/DAEH/ - NDEV,IMAX,IBLOCK,ANHEAD(600)

ARGUMENT LIST:

NDATE - current date
NTIME - current time

INITL SUBPROGRAM

PURPOSE: Sets default value for entire file to -9999

CALLING SEQUENCE: Call INITL

COMMON BLOCKS:

/DAEH/ - NDEV,IMAX,IBLOCK,ANHEAD(600)
/LHEAD/ - LDEV,JMAX,JBLOCK,ALHEAD(400)

NAME SUBPROGRAM

PURPOSE: Returns 28 character name of data base, as stored in node file header (by program NAMEIN)

CALLING SEQUENCE: Call NAME (INAME)

COMMON BLOCKS: None

ARGUMENTS:

INAME - 7 word vector containing character data

NOTES: This module contains an equivalence statment between an integer *4 vector and a real *4 vector.

OPEN SUBPROGRAM

PURPOSE: To open the link and node files, provide the necessary Define File statements, and to initialize the logical update flags to false.

CALLING SEQUENCE: Call OPEN

COMMON BLOCKS:

/UPLINK/ - PIPEBK,NL,UPDATL
/UPNODE/ - VERTBK,NN,UPDATN

NOTES: This module contains the Define File statements for the node and link files.

PUTLK SUBPROGRAM

PURPOSE: To write a logical link record to the link file

CALLING SEQUENCE: Call PUTLK(APIPE,ILK,ISTAT)

COMMON BLOCKS:

/UPLINK/ - PIPEBK,NL,UPDATL
/LHEAD/ - LDEV,JMAX,JBLOCK,ALHEAD(400)

ARGUMENT LIST

APIPE - 40 word link logical record
ILK - link number ID
ISTAT - return code =0 normal return
=1 no insertion ILK out of file range
=2 ILK ≠ APIPE(1)
=3 I/O error

PUTLKZ SUBPROGRAM

PURPOSE: To zero a logical link record in the link file

CALLING SEQUENCE: Call PUTLKZ (AZERO,ILK,ISTAT)

COMMON BLOCKS:

/UPLINK/ - PIPEBK,NL,UPDATL
/LHEAD/ - LDEV,JMAX,JBLOCK,ALHEAD(400)

ARGUMENT LIST:

AZERO - a 40 word vector initialized to zero
ILK - link numeric ID
ISTAT - return code =0 normal
=1 ILK out of file range
=3 I/O error

PUTND SUBPROGRAM

PURPOSE: To write a logical node record to the node file

CALLING SEQUENCE: Call PUTND (ANODE,IND,ISTAT)

COMMON BLOCKS:

/UPNODE/ - VERTBK, NN, UPDATN
/DAEH/ - NDEV, IMAX, IBLOCK, ANHEAD(600)

ARGUMENT LIST:

ANODE - 60 word vector node logical record
IND - node numeric ID
ISTAT - return code =0 normal
=1 IND out of file range
=2 IND ≠ ANODE(1)
=3 I/O error

PUTNDZ SUBPROGRAM

PURPOSE: To zero a logical node record in the node file

CALLING SEQUENCE: Call PUTNDZ (AZERO,IND,ISTAT)

COMMON BLOCKS:

/UPNODE/ - VERTBK,NN,UPDATN
/DAEH/ - NDEV,IMAX,IBLOCK,ANHEAD(600)

ARGUMENT LIST:

AZERO - 60 word vector initialized to zero
IND - node ID of record being deleted
ISTAT - return code =0 normal
=1 IND out of file range
=3 I/O error

UHEAD SUBPROGRAM

PURPOSE: Called in FIXN, used to update link and node file header records.

CALLING SEQUENCE: Call UHEAD (NUMN, NDATE, NTIME,NCODE)

COMMON BLOCKS:

/UBR/ - UBR1
/DAEH/ - NDEV,IMAX,IBLOCK,ANHEAD(600)
/LHEAD/- LDEV,JMAX,JBDCK,ALHEAD(400)

ARGUMENT LIST:

NUMN - highest currently used node numeric ID
NDATE - current date
NTIME - current time
NCODE - editor module update code

ULHEAD SUBPROGRAM

PURPOSE: Used to update header record when an editor module other than FIXN has been run on the files.

CALLING SEQUENCE: Call ULHEAD (NUML,NDATE,NTIME,NCODE)

COMMON BLOCKS:

/DAEH/ - NDEV,IMAX,IBLOCK,ANHEAD(600)
/LHEAD/- LDEV, JMAX, JBLOCK,ALHEAD(400)

ARGUMENT LIST:

NUML - largest currently used link numeric ID
NDATE - current date
NTIME - current time
NCODE - editor module update code

UPDATN SUBPROGRAM

PURPOSE: To insert the topological information into the node file from the link file and return the geographic data through the argument GEO for insertion into the link file.

CALLING SEQUENCE: Call UPDATN (IT,N,ISTART,GEO,ISTAT,IRITE)

COMMON BLOCKS: None

ARGUMENT LIST:

| | | |
|---|---|---|
| IT | - | Numeric link ID for insertion into node connectivity list |
| N | - | Node ID of record to be updated |
| ISTART | - | =0 for 1st node update by link IT<br>=3 for 2nd node update by link IT |
| GEO | - | 6 word vector for coordinate data transfer from node record to link record ISTAT |
| | | = -1 too many links assigned<br>>0 ISTAT return code from I/O subroutine |
| IRITE | - | logical unit on which error messages are to be written |

## DESCRIPTION OF COMMON BLOCKS USED BY SUBPROGRAMS

DAEH
RECT
LHEAD
LUNIT
UBR
UPLINK
UPNODE

COMMON BLOCK REFERENCE INFORMATION

NAME: DAEH

USAGE: /DAEH/NDEV/IMAX,IBLOCK,ANHEAD(600)

VARIABLE DESCRIPTION:

| | | |
|---|---|---|
| NDEV | INTEGER | Logical unit on which to access node file |
| IMAX | INTEGER | Largest logical record number currently used, at program start |
| IBLOCK | INTEGER | Number of physical blocks associated with node file |
| ANHEAD | REAL | 600 word vector. Node file header record at program start |

NAME: RECT

USAGE: /RECT/ XMAX,XMIN,YMAX,YMIN

VARIABLE DESCRIPTION:

| | | |
|---|---|---|
| XMAX | REAL | Maximum easting of file bounding rectangle |
| XMIN | REAL | Minimum easting of file bounding rectangle |
| YMAX | REAL | Maximum northing of file bounding rectangle |
| YMIN | REAL | Minimum northing of file bounding rectangle |

NAME: LHEAD

USAGE: /LHEAD/ LDEV,JMAX,JBLOCK,ALHEAD(400)

VARIABLE DESCRIPTION:

| | | |
|---|---|---|
| LDEV | INTEGER | Logical unit on which to access the link file |
| JMAX | INTEGER | Maximum logical link record, at program start |
| JBLOCK | INTEGER | Number of physical records in link file |
| ALHEAD | REAL | 400 word vector, containing Link file header record at program start |

COMMON BLOCK REFERENCE INFORMATION

NAME: LUNIT

USAGE: /LUNIT/ IREAD,IRITE

VARIABLE DESCRIPTION:

| | | |
|---|---|---|
| IREAD | INTEGER | Logical unit to read input |
| IRITE | INTEGER | Logical unit assigned to printer |

NAME: UBR

USAGE: /UBR/ UBR1

VARIABLE DESCRITION:

| | | |
|---|---|---|
| UBR1 | LOGICAL | Bounding rectangle update flag |

NAME: UPLINK

USAGE: /UPLINK/ PIPEBK,NL,UPDATL

VARIABLE DESCRIPTION:

| | | |
|---|---|---|
| PIPEBK | READ | 400 word array dimensioned (40,10) containing physical record from link file |
| NL | INTEGER | Physical record number corresponding to record in PIPEBK |
| UPDATL | LOGICAL | Physical record update flag |

NAME: UPNODE

USAGE: /UPNODE/ VERTBK, NN, UPDATN

VARIABLE DESCRIPTION:

| | | |
|---|---|---|
| VERTBK | READL | 60 word array dimensioned (60,10) containing Node file physical record |
| NN | INTEGER | Physical record number corresponding to record stored in VERTBK |
| UPDATN | LOGICAL | Physical record update flag |

WATSIM Modifications

The WATSIM program was originally developed for the UNIVAC series of computers, and made use of UNIVAC FORTRAN language. This language is less compatible with ANSI Standard FORTRAN than is IBM FORTRAN. The version of the program used in WSSM is that obtained from the Public Service Company of Colorado. This version is an IBM compatible modification of the original Univac program, and makes use of a single assembler subroutine (IAND) to perform certain bit-masking functions. The IBM version is not otherwise significantly different, in operation or structure, from the original UNIVAC version.

While WATSIM is a comprehensive hydraulic analysis program with many features, and it is in the public domain, the internal coding is neither straight-forward nor well-structured, but is instead quite complex. There are no programmer's guides known to be available for the system.

The primary modifications made to WATSIM relate to obtaining the necessary output, in card-image form, organized by link and node number, for input to WSSM. This modification was complicated by the fact that WATSIM is organized around node numbers, and does not recognize external link numbers - for WATSIM, a link is defined by its two end node numbers. The WSSM, on the other hand, is organized by both explicit node and link numbers. As such, it was necessary to develop a method by which link numbers could be accepted as input to WATSIM, and related to the eventual output. This was accomplished by treating the link numbers simply as labels, to be carried along with no internal computation consequence. That is, link numbers are associated with node pairs only on input and output of data. The process is made more complex due to the internal renumbering algorithms of WATSIM, that

renumber nodes and re-order links in order to get a more efficient simultaneous equation solution. Thus, it is not always obvious within the code of WATSIM whether nodes are being handled in their internal or external numbering.

A further modification to WATSIM involves obtaining summary output for steady-state solutions. In the original, it was impossible to get summary output for steady-state solutions. The program has been modified to provide this output, and the card-image file used for the output interface is taken from the point in the output subroutine at which data is stored for the summary file. The original WATSIM program did not maintain proper signed quantities for flow direction and flow at nodes, and a variety of modifications, again in the output subroutine, have been made to insure that signed quantities transmitted to the output interface are proper. Other minor modifications, associated with initialization of data values, and precision tolerances for calculation, have also been found necessary. The modified program is known to operate properly for steady-state solutions, and should continue to operate in dynamic mode as well, but no testing of dynamic mode operation has been carried out.

In summary, the modifications to WATSIM have been the minimum necessary to support the input and output interfaces. The program is not easily modified, and it is not recommended that 'casual' changes be made to WATSIM.

Transportation Issues

The WSSM computer code is programmed in "near-standazrd" FORTRAN IV, and should not present any major problems in terms of transportability. As noted previously, the WATSIM code has shown minor problems in terms of its transportability. The version of WATSIM used with the WSSM in its implementation on the University of Michigan Terminal System contains an assember-language subroutine, IAND, used for character manipulation, which will need to be changed on non-IBM compatible computers.

Many of the programs of WSSM make use of the DATIME subroutine, to provide the data and time from the computer operating system, for purposes of identification and tracking on printed output and within the file header. DATIME is installation-specific, and will need to be replaced by an appropriate subroutine, delivering the date and time in appropriate format, as indicated in the subprogram description for subroutine DATIME, in this report. If the operating system at the host computer cannot deliver date and time, then a dummy subroutine must be created, delivering constant values when called.

As the WSSM make use of direct-access programming, the conventions of which vary from computer to computer, modifications may be required for those subprograms containing the DEFINE FILE and direct-access I/O statements. Almost all such direct-access references within the WSSM code have been localized to these input/output routines, to minimize transportability problems.

The final major transportability issue relates to the graphics software utilized by program NETPLOT. This program makes use of so-called "industry-standard" Calcomp-format subroutine calls for the basic plotting functions. Most other plotter manufacturers have adopted the same subroutine calling conventions as Calcomp. These basic plotting routines are proprietary to

each graphic device manufacturer, and thus cannot be distributed as a portion of the WSSM, but must be available at the host installation. Of the plotting routines used by NETPLOT, all are standard "basic" plotting routines (i.e normally available in the minimum complement of plot subroutines provided by a plotter manufacturer) with the exception of AROHD, used to draw the arrows showing flow direction for pipes, which is normally made available by Calcomp as a portion of a more advanced graphic routines package. Thus, a compatible AROHD routine must be available, or must be created at the user installation. Creation of such a routine should be fairly simple, if only a single format of arrow-head drawing is desired.

It is recommended that, at any given WSSM installation, subroutines be maintained in a library in load module form, such that they can be linked in to main programs at run-time. WSSM is organized so as to make use of a variety of standard utility sub-programs, and thus lends itself to this form of approach. Accordingly, in bringing up a WSSM installation, it is suggested that the subroutines first be compiled into a subroutine library. Any incompatabilities uncovered during the compilation process will be the clearest indication of transportability problems that may be unique to a particular installation.

## REFERENCES


1. Clark, R. M., and Males, R. M., "Water Supply Simulation Model, Volume I - Model Development", USEPA, Cincinnati, OH, May 1982.

2. Clark, Robert M. and Stevie, Richard, Water Supply Simulation Model, Volume II - Model Development, Drinking Water Research Division, Municipal Environmental Research Laboratory, USEPA, Cincinnati, OH 45268, Unpublished.

3. Rao, H. S. Bree, D. W. Sr., Benzri, R., "Extended Period Simulation of Water Distribution Networks", February 1974, National Technical Information Service Report, PB 230 148.

4. WATER User's Manual, University of Waterloo, Waterloo Research Institute, Waterloo, Ontario, Canada.